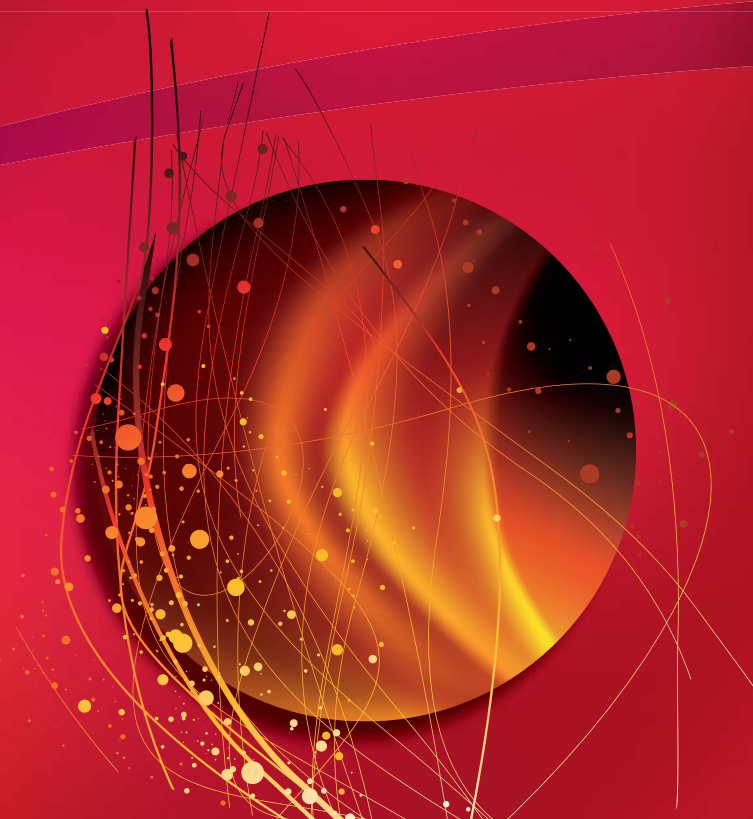


Le Campus

ActionScript 3 et le motion design



 Codes sources
sur www.pearson.fr

Arzhur Caouissin

PEARSON

ActionScript 3 et le motion design

Pearson Education France a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, Pearson Education France n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Pearson Education France ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou autres marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par Pearson Education France

47 bis, rue des Vinaigriers

75010 PARIS

Tél. : 01 72 74 90 00

www.pearson.fr

Avec la collaboration de Bruno Sébarbe

Éditrice : Dominique Buraud. Digit Books

Mise en pages : TyPAO

ISBN : 978-2-7440-4128-0

Copyright © 2010 Pearson Education France

Tous droits réservés

Aucune représentation ou reproduction, même partielle, autre que celles prévues à l'article L. 122-5 2° et 3° a) du code de la propriété intellectuelle ne peut être faite sans l'autorisation expresse de Pearson Education France ou, le cas échéant, sans le respect des modalités prévues à l'article L. 122-10 dudit code.

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

ActionScript 3 et le motion design

Arzhur Caouissin

PEARSON

Table des matières

Préface	1
À qui s'adresse ce livre	2
Les intentions de l'auteur	2
Structure du livre	3
Les exemples	5
Conventions	5
Remerciements	6
Présentation de l'auteur	6
1 Les animations en ActionScript	7
Introduction	7
Animer les propriétés : position, dimension, rotation, opacité et échelle	7
Gérer les propriétés avec des conditions	14
Gérer les accélérations	18
Gérer le défilement de panoramiques en boucle	23
Synthèse	28
2 Interpolations et interactivité avec les classes Tween et TweenMax	29
Introduction	29
Animer une galerie avec la classe Tween intégrée	29
Animer une carte interactive avec la classe TweenMax	38
Défilant TweenMax avec un pointeur et un masque	48
Synthèse	56

3 Les transitions d'effets et de filtres	57
Introduction	57
Galerie photo avec transition d'effets	57
Effet de fondu avec un Timer	57
Définition de la classe <i>transitionManager</i>	62
Galerie photo avec transition de filtres	65
Filtre flou	68
Filtre ombre portée	68
Filtre halo	69
Filtre biseau	70
Déclinaison des filtres	71
Synthèse	75
4 La programmation de squelettes	77
Introduction	77
Programmer un mouvement mécanique	77
Définition du squelette	81
Animer le squelette	84
Programmer un mouvement organique	86
Basculer du mode animation programmée au mode exécution	92
Activer un squelette chargé dans un SWF	94
Synthèse	97
5 Les galeries d'images	99
Introduction	99
Afficher des images externalisées et aléatoires	99
Réaliser une jauge de chargement	103
Réaliser une galerie d'images externalisées	107
Réaliser une galerie d'images avec XML	113
Interactivité sur les objets dynamiques	122
Résoudre les erreurs éventuelles au chargement et à l'exécution	131
Gestion de site dynamique avec XML	134
Synthèse	135

6 La vidéo standard et composite en FLV	137
Introduction	137
Concevoir la vidéo pour le Web	138
Composition simple avec Apple Motion	141
Composition simple avec Adobe After Effects	147
Échantillonner la vidéo pour Flash	153
Encoder en FLV avec After Effects	154
Encoder en FLV avec Premiere Pro	154
Encoder en FLV avec Final Cut Pro	154
Encoder en AVI-DV avec Window Movie Maker	154
Encoder en MOV ou DV avec iMovie	155
Encoder en FLV avec Adobe Media Encoder	155
Intégrer de la vidéo composite dans Flash	169
Synthèse	175
7 La vidéo HD en F4V	177
Introduction	177
Encoder en F4V avec Adobe Media Encoder	177
Onglet Vidéo	179
Onglet Audio	183
Encodage F4V avec Quick Time	184
Créer un lecteur vidéo personnalisé	188
Créer un lecteur vidéo H-264 pour Flash 6 et plus	191
Agrandir une vidéo sans perte	197
Synthèse	199
8 La vidéo interactive	201
Introduction	201
Contrôles de base de la vidéo	201
Lecture automatique	203

Mise en cache	203
Lire la vidéo	204
Arrêt de la vidéo	204
Accélérer la vidéo	204
Rembobiner la vidéo	205
Modifier le volume audio	205
Augmenter et diminuer progressivement le son	206
Chapitrage vidéo	207
Sous-titrage vidéo	211
Boucle vidéo	215
Synchroniser des actions avec les points de repère	217
Repères de navigation	218
Repères d'événements	223
Lire une vidéo en arrière	227
Arrêter une vidéo dans un SWF imbriqué	232
Arrêt et fermeture depuis le document racine	234
Arrêt et fermeture depuis le contenu chargé	236
Synthèse	239
9 La 3D native	241
Introduction	241
Animer un livre 3D en ActionScript	242
Carrousel d'images 3D	251
Version simplifiée	252
Version dynamique	256
Mur d'images 3D	260
Galerie vidéo 3D circulaire	268
La galerie simple	270
La galerie optimisée	276
Navigation spatiale 3D façon TimeMachine ou Aero	279
Synthèse	287

10 La 3D et <i>PaperVision</i>	289
Introduction	289
Installer <i>PaperVision</i>	289
Téléchargement avec Tortoise SVN pour Windows	290
Téléchargement avec SVN X pour Mac OS X	292
Intégrer la classe <i>PaperVision</i> à Flash	297
Modéliser en 3D pour Flash et <i>PaperVision</i>	299
Programmer les mouvements de caméra 3D	304
Programmer les mouvements d'objets 3D	311
Interactivité avec les touches du clavier	315
Synthèse	329
 11 API d'affichage et de colorimétrie	 331
Introduction	331
Lissage des images bitmap	331
Lissage des graphismes vectoriels	335
Effet loupe avec optique déformante	338
Filtres de correction colorimétrique	343
Appliquer et restaurer un filtre	343
Correction colorimétrique par lot	349
Correction colorimétrique par interpolation	351
Imprimer un document SWF	353
Appliquer une teinte aléatoire	355
Créer un puits de couleurs	357
Synthèse	360
 12 Le Web en vrai relief	 361
Introduction	361
Prise de vues pour le relief	362

Réaliser un anaglyphe avec Photoshop	363
Le principe de l'image jaillissante ou fenêtre	363
Créer l'anaglyphe à partir de deux images	370
Gérer un anaglyphe en ActionScript	375
Interface SWF en relief dynamique	377
Synthèse	387
13 Les systèmes de navigation avancés	389
Introduction	389
Boutons MovieClip fixes avec état activé et visité	389
Les boutons visités	389
La structure <i>Switch... case</i>	393
Les boutons activés	394
Boutons MovieClip animés et interfaçables	395
Méthode avec les actions dans le scénario du lien	397
Méthode avec les actions sur la scène principale	401
Console de navigation en miniature	404
Menu déroulant à repositionnement automatique	407
Activer et désactiver les boutons et les MovieClip	414
Synthèse	415
14 La communication Flash/HTML	417
Introduction	417
Menu contextuel du lecteur Flash	417
Navigation Flash vers le HTML	420
Navigation HTML vers Flash	423
Historique de l'animation Flash dans le navigateur	428
Importer du HTML dans un SWF	430
Gérer le Flash transparent	434
Synthèse	435

15 La gestion de sites Full Flash	437
Introduction	437
Basculer en mode plein écran et restaurer l’affichage standard	437
Interface élastique flottante	443
Gérer le PDF	450
Gérer la typographie	451
Les textes statiques	452
Les textes dynamiques	452
Les bibliothèques partagées de symboles et de typographies	454
Importer Flash AS1/AS2 dans Flash AS3	458
Synthèse	462
 16 Les solutions de référencement du Flash	 463
Introduction	463
Intégration XHTML stricte	463
Structurer un document pour l’API de Google	464
Les métadonnées des fichiers SWF	467
Gérer le contenu alternatif pour les robots et les appareils nomades	469
Installer le kit SWFObject2	469
Personnaliser le document	472
Synthèse	476
 17 L’accessibilité dans Flash	 477
Introduction	477
La fenêtre Accessibilité	478
Concevoir un document Flash pour les non-voyants et malentendants	481
Synthèse	482
 18 Ressources	 483
Livres	483

Tutoriels vidéos	483
Sites web et forums	484
Dictionnaires	484
Index	485
Index des notes et encadrés	485



Préface

Le motion design se définit par l'ensemble des signes fixes et animés qui caractérisent l'identité d'un site et au service de son contenu. Ces signes et ces formes véhiculent une identité par des animations distribuées dans un scénario sur une échelle de temps linéaire. Par cette organisation plastique et sémantique, ils en appuient le sens dégagé.

Pour offrir une identité qui puisse évoluer selon la configuration et le comportement de l'utilisateur, une identité évolutive, réactive et applicative, il est incontournable de la réaliser en ActionScript. Nous parlons alors de motion design en ActionScript. Nous pourrions parler aussi de script design, mais le terme n'est pas vraiment usuel.

Pour cela, nous disposons de plusieurs versions du langage ActionScript. Il convient naturellement d'aborder la version qui offre la réponse la plus appropriée au besoin défini par le concepteur du site. Trois versions de ce langage sont apparues successivement depuis la fin des années 90 et continuent d'exister chacune indépendamment des autres :

- ActionScript 1.0 est utilisé pour le Web mobile et les sites de première génération. Il n'est presque plus employé pour la création de sites Internet classiques car il est trop restreint en fonctionnalités. Son lecteur est léger d'où l'intérêt conservé pour sa portabilité.
- ActionScript 2.0 offre une palette de comportements bien plus étoffée et demeure encore largement employé pour bon nombre des créations de sites dits vitrines, n'exprimant pas un grand besoin de réactivité ni de fonctionnalités. Son apprentissage est encore relativement accessible et son lecteur est déjà déployé sur de nombreux appareils mobiles.
- ActionScript 3.0, apparu depuis Flash CS3 (Flash 9), permet de réaliser des applications complexes avec une grande réactivité. Concernant la partie graphique, elle autorise principalement la 3D et la gestion de la vidéo en haute définition, ainsi que le développement d'applicatifs lourds comme le traitement des images, gourmands en calculs. Si le langage semble plus complexe au premier abord, il n'en est rien dans la pratique, car en réalité, la multiplication des lignes de code requise en AS3 vient de la nécessité de détailler le contexte d'utilisation de chaque action. Ceci afin de réserver l'espace mémoire uniquement requis pour les animations, et donc, d'en améliorer les performances à l'affichage. Mais les actions elles-mêmes restent souvent équivalentes à celles déjà utilisées en AS2. D'autres modifications d'ordre structurel ont, cela dit, été apportées en AS3, qu'il est nécessaire d'apprivoiser, surtout lorsque l'on a acquis des habitudes de développement tranquilles, encore possibles en AS2.

Ce livre a pour objectif de vous permettre d'acquérir facilement les notions d'ActionScript 3 nécessaires pour vous rendre indépendant dans le développement d'interfaces riches et graphiques, à partir de symboles placés dans le scénario. Il vous permettra aussi de contourner les problèmes éventuellement rencontrés suite aux modifications structurelles induites par ce nouveau langage, notamment si vous utilisiez déjà AS2 et aviez pris

l'habitude de dialoguer facilement d'un document SWF à un autre. Ainsi, en plus des principales fonctionnalités déjà évoquées comme la 3D et la vidéo en haute définition, vous apprendrez à imbriquer des animations SWF entre elles, mais en sachant réellement les faire interagir l'une envers l'autre. Vous apprendrez également à afficher et masquer des informations, mais en connaissant comment désactiver les contenus supprimés alors qu'ils demeurent intrinsèquement encore actifs. Vous verrez autant d'autres principes inhérents à AS3 qu'un ouvrage plus scolaire ne vous aurait pas permis de contrôler et qu'un livre trop technique vous aurait découragé de découvrir.

Cet ouvrage, où nous abordons les spécificités graphiques d'AS3 et quelques principes déjà présents avec AS2, peut être appréhendé ponctuellement, au cas par cas, pour résoudre des besoins ciblés, ou bien dans son intégralité, pour mieux englober l'ensemble de la valeur ajoutée que constitue désormais, pour tout concepteur graphique, ce nouveau langage. Avec ce livre, vous serez en mesure, je le souhaite, de réaliser vous-même – et facilement – les sites les plus aboutis graphiquement et aux fonctionnalités avancées.

À qui s'adresse ce livre

Nous détaillons dans cet ouvrage les techniques utilisées en production par les webdesigners. Ce livre est destiné aux flasheurs débutants qui souhaitent contrôler les contenus depuis le scénario, en codant le plus simplement possible de vraies fonctionnalités.

Tout graphiste flasheur débutant, en poste, en freelance, ou en étude, peut trouver dans cet ouvrage les réponses aux questions qu'il se pose afin de maîtriser ActionScript 3, directement dans le scénario de Flash CS4.

Toute personne intéressée par le déploiement de contenus enrichis avec Flash, maîtrisant déjà cette technique, mais inspirée par des notions inédites, comme le référencement, l'accessibilité, le relief, l'importation d'objets 3D, entre autres, trouvera également des réponses sur ces sujets.

Pour aborder ce livre, vous devez seulement maîtriser quelques rudiments du langage ActionScript 3 : le placement du code dans le scénario, la gestion des fonctions et des écouteurs, ainsi que la navigation dans le scénario.

Les nombreux exemples commentés de ce livre en font un guide pratique incontournable pour toute personne néophyte dans la programmation en ActionScript 3 et le motion design.

Les intentions de l'auteur

Pour avoir été longtemps graphiste indépendant sur les nouvelles technologies, dès leur apparition dans les années 90, je comprends le besoin de mes amis designers et de mes élèves graphistes de mettre en forme les contenus efficacement et facilement.

Pour la communauté de concepteurs, de designers, de créatifs, à laquelle j'appartiens, la problématique est de ne pas consacrer son précieux temps à seulement coder. L'énergie déployée en production doit privilégier avant tout la conception et la créativité.

Si Flash est désormais un brillant outil pour l'animation et si ActionScript 3 est désormais un puissant langage pour l'interactivité, qu'en est-il de la programmation au service de la

création ? À la question peut-on programmer de l'interactivité avancée sur des contenus graphiques placés à même le scénario, ma réponse est oui ! Et comment !

Depuis que ce langage est arrivé, aucun ouvrage n'a jusqu'ici su apporter ce qui était réellement utile pour un designer. Aucun ouvrage n'a su présenter de manière accessible et avancée, les techniques pour organiser dans le scénario tous les médias et tous les types d'interactions possibles entre ces différents médias, et ceci avec des exemples concrets et efficacement. Une idée pourtant simple, mais devant laquelle un désert pédagogique m'a laissé pantois.

Oui, le ciblage est autorisé et possible entre SWF imbriqués. Oui, la 3D est facilement contrôlable depuis le scénario. Oui, une ligne de code suffit pour créer des animations époustouflantes. Oui, la vidéo en F4V et les points de repère sont compatibles, y compris avec d'anciens lecteurs Flash. Oui, nous pouvons même, avec Flash, expérimenter des choses incroyables comme le relief, le référencement et l'accessibilité.

Face aux innombrables questions que mes élèves se sont posées lors de leur découverte d'ActionScript 3, à celles auxquelles se sont heurtés mes collègues en production, astreints depuis quelques années à la création d'interfaces Flash sans la moindre interactivité, ainsi que mes amis designers ulcérés (oui, oui, ulcérés) par l'invraisemblable mécanique de ce langage, j'ai souhaité apporter mes réponses.

Il existe deux approches, souvent mises en opposition, pour appréhender l'ActionScript 3. La première consiste à tout coder de manière stricte dans des classes externes et ne rien introduire dans l'interface auteur ni sur le scénario. Cette méthode permet d'optimiser la vitesse d'exécution d'une animation de quelques micro-secondes. Mais lorsque nous considérons que la valeur ajoutée doit davantage être focalisée sur l'organisation du contenu et sa forme, plutôt que sur ce temps d'exécution, nous ne pouvons accepter de devoir programmer des pages entières de code, et y consacrer bien trop de temps, au détriment de la créativité et... de sa santé.

À défaut de pouvoir déléguer une partie de la programmation à un tiers, l'autre choix consiste donc à placer, directement dans le scénario, les seules instructions utiles, pour mettre en forme les contenus, sans compromettre cette noble créativité et sans corrompre non plus les avancées du langage ni les performances de l'application.

Devant cette nécessité de devoir livrer toutes mes astuces, j'ai donc choisi de proposer un ouvrage qui rassemble de manière claire, accessible et évidente, ce que j'aurais moi-même jubilé de trouver si je débute : un livre synthétique sur ce que l'on pourrait nommer ActionScript 3 et le motion design.

Structure du livre

Cet ouvrage présente les instructions ActionScript 3 nécessaires pour développer des concepts inhérents à la création interactive avec Flash CS4. Dans ce livre, nous détaillons les points suivants :

- L'animation réalisée à partir d'instructions simples en ActionScript : les accélérations et les boucles.

- La création d'animations à partir de classes importées afin de permettre la création d'interfaces animées entièrement dynamiques et modifiables selon le comportement de l'utilisateur.
- L'animation de filtres et d'effets appliqués à des contenus graphiques.
- La gestion des squelettes avec ActionScript afin de permettre le contrôle dynamique de l'animation de personnages ou de structures mécaniques ainsi que le mouvement de formes organiques composées à partir de graphismes vectoriels.
- La création de galeries d'images simples et plus avancées, avec et sans XML, ainsi que l'interaction avec des objets placés dynamiquement dans la scène.
- La gestion de la vidéo au sein de Flash, à partir du format FLV, et la gestion de vidéos composites avec la transparence pour l'implémentation d'effets spéciaux.
- La gestion de vidéos en haute définition, avec le format F4V et le codec H-264, à partir de séquences réalisées par la société gKaster, et compatibles avec d'anciens lecteurs Flash.
- L'interactivité en vidéo avec les contrôles de lecture, le chapitrage, les points de repère, mais surtout toute la complexité liée à une gestion fluide et optimale du signal vidéo, avec lecture en arrière et arrêt complet de la vidéo dans un document embarqué.
- La 3D native disponible dans Flash, avec la création d'interfaces graphiques (murs d'images, galeries vidéo, navigations spatiales).
- La modélisation et l'animation d'objets 3D réels, avec le logiciel gratuit Google Sketchup et la classe libre PaperVision.
- L'API de traitement de l'image avec la gestion dynamique de filtres colorimétriques afin de permettre la retouche des images et le lissage des contenus déformés, pivotés ou zoomés.
- Le relief avec la création d'un contenu en relief depuis Photoshop, et son intégration dans Flash. Nous abordons aussi dans ce chapitre la réalisation inédite d'une interface dynamique qui affiche, automatiquement en relief, tout contenu distribué sur un index Z de profondeur 3D.
- La création de systèmes de navigation avancés avec la résolution de problèmes et de contraintes souvent observées en production, comme la création de boutons à états visités et les boutons interfaçables imbriqués.
- La communication entre un document HTML et un document Flash, afin d'améliorer la relation entre les contenus dans une interface de site hybride.
- La gestion de sites tout en Flash avec la réorganisation des contenus de manière à créer des interfaces élastiques, mais non déformantes, et disponibles pour un affichage en plein écran.
- Les connaissances requises pour la création d'un site en Flash, accessible pour les moteurs de recherche, et l'affichage de contenu alternatif pour les utilisateurs qui ne disposent pas du lecteur Flash.
- En annexe, vous trouverez les outils de développement qui permettent de rendre un document Flash accessible aux personnes à mobilité réduite sur le Web, ainsi que des références et aides pour compléter votre apprentissage. Enfin, un index classique ainsi qu'un index des notes et remarques complètent cet ouvrage.

Les exemples

Chaque étape du livre est présentée avec une logique pédagogique progressive. Chacun des chapitres aborde un thème spécifique et fait référence à des exemples disponibles en Flash au format natif (.fla). Vous pouvez utiliser ces fichiers et leur code, librement, à l'exception des vidéos et des images qui y sont placées uniquement pour illustrer l'ouvrage.

L'ensemble des exemples du livre est disponible en téléchargement à l'adresse suivante : <http://www.pearson.fr/livre/?GCOI=27440100685160>.

Pour suivre confortablement les exemples commentés du livre, veuillez télécharger distinctement les différents dossiers compressés mis à votre disposition, puis rassembler leurs contenus respectifs dans le même répertoire que vous créerez et nommerez "Exemples". Tout au long du livre, nous faisons référence aux fichiers placés dans ce dossier "Exemples". Aussi, en organisant vos téléchargements de la sorte, vous en faciliterez la lecture.

Dans les exemples, de nombreux fichiers font également référence à des contenus placés dans des répertoires ou des classes ActionScript et JavaScript. Pour votre confort, il convient donc de télécharger l'ensemble des dossiers compressés et de tous les rassembler dans ce nouveau dossier "Exemples" que vous aurez créé. Vous pourrez alors exploiter librement les fichiers appelés tout au long de la lecture de l'ouvrage.

Conventions

Ce livre contient quelques morceaux de code, représentés avec cette convention :

```
//----- actions
this.addEventListener(Event.ENTER_FRAME,defilementPanoramique);
```

Dans le corps du texte, les éléments relatifs à ActionScript sont notés avec cette typographie.



Ce pictogramme signale que vous pouvez télécharger les fichiers d'exemple sur le site de Pearson.

Nous disposons d'autre part de deux types d'encadrés :



Ces encadrés sont des remarques : il peut s'agir d'un point auquel faire attention, d'une référence externe, d'une définition, d'une astuce, d'un renvoi...

Encadrés

Ces encadrés sont des encadrés de référence. Ils émaillent le texte et précisent certains points de manière formelle. Ils sont recensés dans la Table des encadrés en fin d'ouvrage.

Remerciements

Naturellement, je tiens à remercier mon éditeur pour m'avoir accompagné avec un réel dévouement et un grand enthousiasme dans ce projet (Dominique Buraud principalement et Patricia Moncorgé). Je remercie bien sûr Bruno Sébarte pour son implication, ses compétences techniques et ses recommandations toujours constructives. Prends bien soin !

Je remercie également toute l'équipe des Gobelins et Pyramyd pour leur accompagnement et leur soutien sans cesse renouvelé depuis mes débuts (Marie-France, Didier, Sylvie, Sandrine, Anne Tord, Jean-Marie) ; Olivier Coach pour ses conseils avisés sur le relief ; la société GKaster pour leur généreuse mise à disposition de ressources vidéo graphiques ; Thibaut Imbert, Anne Tasso et David Tardiveau pour leurs attentions respectives ; mes élèves et ma clientèle pour l'expérience que nous avons tissée ensemble ; et l'ensemble des graphistes flasheurs développeurs pour qui j'ai conçu ce livre et pour qui la création implique nécessairement un dépassement de soi, dépassement que j'espère enfin possible à mon humble niveau grâce à ce manuel. Je remercie enfin mes proches pour leur patience car l'élaboration d'un tel ouvrage prend du temps et de l'énergie.

Présentation de l'auteur

Arzhur CAOUISSIN est auteur et réalisateur multimédia indépendant, affilié AGESEA. Déjà auteur de trois livres et de nombreux supports pédagogiques au format numérique, il est aussi formateur depuis plus de dix ans en région parisienne, à l'école des Gobelins et chez Pyramyd. Il a également été formateur occasionnel pour l'INA, le GRETA Réseau graphique de l'école Estienne, l'université de Marne La Vallée, la CNAM, l'ILOI et l'école supérieure de design Olivier de Serres.

Spécialisé dans la technologie Flash, il est aussi consulté pour sa vision transversale et artistique sur les différents médias, dans la mode, les cosmétiques, la finance, mais aussi par des agences, pour son expertise en création de contenus web enrichis. Proche du milieu de l'animation, il est intervenu enfin en tant qu'assistant à la mise en scène et à l'écriture, pour des projets filmiques courts, impliquant, entre autres, des captations en studio sur fond vert et le traitement vidéo y compris de scène animées, avant leur intégration dans des univers multimédia.

1

Les animations en ActionScript

Introduction

Les animations réalisées en ActionScript, qu'elles constituent des défilements panoramiques ou des éléments de décor simplement mobiles, permettent de mettre en scène des contenus sans recourir à l'utilisation du scénario. L'avantage de ce procédé, par rapport au scénario, est que l'on peut facilement modifier des valeurs (positions, échelles, angles de rotation) en fonction d'une variable sans avoir à recréer toute l'animation. Ce procédé n'étant pas exclusif, vous pouvez parfaitement cumuler l'avantage d'une animation élaborée, réalisée dans le scénario, à la souplesse d'une animation codée en ActionScript.

Dans les exemples qui suivent, vous allez apprendre à animer des mises en forme avec ActionScript, à redéfinir les propriétés des objets de la scène avec un amortissement, et à gérer des défilements en boucle dans un environnement en perpétuel mouvement. À l'issue de ce chapitre, vous serez en mesure de vous abstraire du scénario pour réaliser une partie de vos animations en code, et vous permettre, en plus de la souplesse obtenue dans la conception des animations, d'alléger déjà la structure du document.

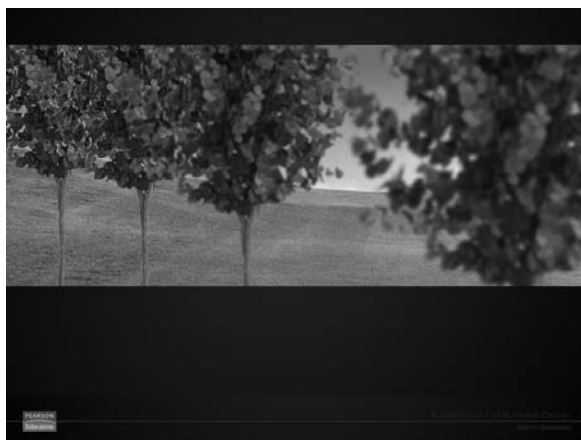
Animer les propriétés : position, dimension, rotation, opacité et échelle

L'exemple que nous proposons ici est un décor multi-plans composé de clips distincts. Chaque clip est déplacé en incrémentant ses propriétés dans un gestionnaire de type `ENTER_FRAME`.

Ce décor, une fois animé, représente une animation de type panoramique qui disparaît progressivement pour laisser place à l'arrière-plan du document principal (voir Figure 1.1).

Figure 1.1

Aperçu de l'exercice.



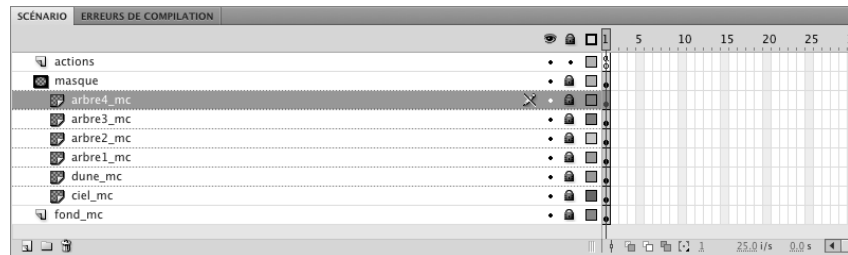


Exemples > ch1_animationEnActionScript_1 fla

Le document que nous utilisons présente la structure suivante : dans le scénario de la scène principale, au-dessus du calque `fond_mc`, clairement répartis vers des calques distincts et masqués, nous pouvons accéder à différents symboles de type `MovieClip` ayant tous un centre placé en haut et à gauche. Chacun de ces symboles dispose, en outre, d'un nom d'occurrence qui le distingue bien des autres et lui permet d'être contrôlé par ActionScript. Pour faciliter l'identification de l'objet depuis la fenêtre Actions, le nom d'occurrence a été répété sur le calque qui contient chaque occurrence de symbole (voir Figure 1.2).

Figure 1.2

Structure du document.



Nommer les calques automatiquement. Pour renommer automatiquement chacun des calques qui contient une occurrence de symbole, plutôt que de nommer chaque calque manuellement, pensez à répartir la ou les occurrences de symboles vers les calques par un clic droit ou `Ctrl+clic > Répartir vers les calques`, directement sur la ou les occurrences de symboles disponibles sur la scène. Attention toutefois, si une occurrence de symbole ne possède pas d'identifiant, c'est le nom du symbole, disponible dans la bibliothèque, qui sera utilisé par Flash pour nommer chacun des nouveaux calques créés.

En haut du scénario, un calque nommé `actions` contient le code suivant :

```
//----- initialisation

var vitesseDefilement:Number=5;

//----- actions

this.addEventListener(Event.ENTER_FRAME,defilementPanoramique);

function defilementPanoramique (evt:Event) {
    //////////// x
    ciel_mc.x-=vitesseDefilement;
    dune_mc.x-=vitesseDefilement*0.2;
    arbre1_mc.x-=vitesseDefilement*0.3;
    arbre2_mc.x+=vitesseDefilement*0.5;
    arbre3_mc.x+=vitesseDefilement*0.8;
    arbre4_mc.x+=vitesseDefilement*0.9;

    //////////// y
    ciel_mc.y-=1;

    //////////// width et height
    ciel_mc.width+=10;
    ciel_mc.height+=10;
```

```

////////// rotation
ciel_mc.rotation-=0.1;

////////// alpha
ciel_mc.alpha-=0.01;
dune_mc.alpha-=0.01;
arbre1_mc.alpha-=0.025;
arbre2_mc.alpha-=0.03;
arbre3_mc.alpha-=0.035;
arbre4_mc.alpha-=0.04;

////////// scaleX et scaleY
dune_mc.scaleX+=0.001;
dune_mc.scaleY+=0.001;

// trace
trace(ciel_mc.x);

if (ciel_mc.x<-600) {
    this.removeEventListener(Event.ENTER_FRAME,defilementPanoramique);
}
}

```

Ce code est structuré en deux parties. Une partie énumère les valeurs à configurer avant de lancer les actions. L'autre partie affiche les programmes à exécuter à partir de ces valeurs préalablement initialisées. Le programme utilisé pour le défilement du panoramique utilise la variable `vitesseDefilement` pour déterminer le positionnement de chaque objet. Mais, c'est selon la position de l'objet `ciel_mc` que nous interrompons ou non l'exécution de l'animation. Le positionnement des objets est donc indirectement arbitré selon la position de `ciel_mc`.

Philosophie d'AS3

Nous entrons ici dans le cœur de la philosophie d'ActionScript 3 où tout doit être clairement défini et identifié par le lecteur Flash avant de pouvoir exécuter les programmes développés. L'objectif de la déclaration préalable des éléments utilisés est que, ce faisant, nous prévenons le lecteur Flash de l'espace mémoire qu'il doit réserver pour chaque objet déclaré avant de les utiliser.

Ainsi, lorsqu'ils sont manipulés au moment des actions du programme, l'exécution des scripts est accélérée car seul l'espace requis par ces objets est déployé par le lecteur. C'est la raison pour laquelle les lignes de commande en AS3 sont plus nombreuses qu'en AS2, mais concourent à un meilleur résultat lors de l'exécution du document par le lecteur Flash car l'animation est alors optimisée.

Tout d'abord, nous déclarons les valeurs à utiliser :

```
var vitesseDefilement:Number=5;
```

Dans la partie initialisation, une valeur de type nombre, `vitesseDefilement`, représente le coefficient d'accélération du défilement panoramique. Plus cette valeur est élevée, plus le défilement sera accéléré.

Puis, nous développons le programme :

```

//----- actions
this.addEventListener(Event.ENTER_FRAME,defilementPanoramique);

```

Tout d'abord, nous attachons un écouteur sur l'image courante avec un gestionnaire de type `Event.ENTER_FRAME` qui permet d'exécuter une fonction au rythme exact de la cadence de la scène. À raison d'une cadence de 25 ips, la fonction sera exécutée 25 fois chaque seconde. Cette fréquence est suffisante pour simuler une progression continue et homogène de valeurs, telle que la modification de la position d'éléments dans notre décor, par exemple. La fonction appelée en deuxième paramètre est identifiée par le terme `defilementPanoramique` et développée plus loin dans le code.

Le langage

Nommer une fonction

Le nom attribué à une fonction est libre, mais ne doit pas comporter d'espace, ni de caractères accentués ou spéciaux. Pour faciliter l'identification de ce que contient la fonction, il est souhaitable de la nommer de manière intelligible. Par exemple, pour une fonction destinée à restaurer la position d'un menu déroulant, nous pouvons la nommer `restaurationMenuDeroulant`. La syntaxe ici employée pour nommer la fonction est la syntaxe dite chameau (*Camel*, en anglais).

Syntaxe chameau et séparateurs

La syntaxe chameau (ou *Camel*) consiste à nommer un objet, une variable ou une fonction, en collant les mots que vous avez choisis arbitrairement et qui composent son nom, en les séparant à l'aide d'une majuscule. Par exemple, la désignation `lancerLeMenuDeroulant` qui peut identifier une fonction destinée à déployer un menu déroulant, utilise la syntaxe chameau. Cette syntaxe permet d'éviter d'utiliser le séparateur tiret (-) considéré en ActionScript comme un opérateur de soustraction. L'expression `lancer-le-menu-deroulant` aurait en effet indiqué au lecteur Flash de soustraire la valeur déroulant à la valeur menu, elle-même soustraite à la valeur 1e, elle-même soustraite à la valeur lancer.

Nommer les occurrences

La syntaxe chameau est également utilisée pour nommer les occurrences de symbole. On y ajoute, en revanche, pour faciliter l'identification du type de l'objet lors de son apparition dans le code, une pseudo extension. Une pseudo extension permet de signaler la nature de l'objet (`MovieClip`, bouton, etc.) lorsque celui-ci est mentionné dans le code ActionScript sans avoir à revenir sur l'objet qui figure dans la bibliothèque ou dans la scène. Pour créer une pseudo extension, qui ne soit pas perçue comme un opérateur par le lecteur Flash, nous utilisons le caractère underscore (_) suivi de caractères qui définissent l'objet créé. Par exemple : `menu_mc`, `lien_btn`, `legende_txt` et `ecran_video` utilisent les extensions les plus couramment employées et désignent respectivement un `MovieClip`, un bouton, un champ de texte dynamique et un composant vidéo. Vous pouvez créer vos propres extensions, mais celles qui sont présentées ici sont reconnues pour être utilisées par un grand nombre de flasheurs et elles vous aideront dans le cadre de développements collaboratifs. Une occurrence de symbole ne peut en outre pas être composée de séparateur tiret (-) toujours considéré comme un opérateur de soustraction, ni commencer par un chiffre.

Plus loin, dans notre programme, nous créons la fonction `defilementPanoramique` appelée par le gestionnaire d'événements. Cette fonction reprend le nom de la classe `Event`, en paramètre, utilisée dans le gestionnaire et ne peut ainsi être exécutée que par ce type de gestionnaire. La syntaxe du terme `evt` qui précède le rappel de la classe employée reste à votre appréciation. Vous pourriez aussi bien le nommer `titi` ou `Balthazar`, cela n'affecte en rien le code.

Il s'agit simplement d'utiliser un nom qui permette de faire référence, plus tard dans la fonction, à la classe utilisée, comme le ferait une variable dans un autre contexte :

```
function defilementPanoramique (evt:Event) {
}
```

Usuellement, nous rencontrons aussi les termes `e`, `event`, `evenement` ou `EventObject`, pour désigner la classe véhiculée dans la fonction. Une fonction doit toujours véhiculer la classe invoquée par le gestionnaire d'événements qui s'y réfère (par exemple : `MouseEvent`), car un gestionnaire d'événements n'est autre qu'une méthode qui renvoie, à la fonction appelée, le paramètre événement auquel justement il se rattache. La fonction appelée par le gestionnaire doit donc être typée pour recevoir ce paramètre et pouvoir être exécutée normalement. C'est la raison pour laquelle vous ne pouvez pas utiliser de fonction autonome (c'est-à-dire, sans typage d'événement), si celle-ci est appelée directement par un gestionnaire d'événements. Nous verrons, plus loin dans cet ouvrage, qu'il est en revanche possible d'appeler une fonction autonome à partir d'une fonction, qui, elle, sera belle et bien typée.

Les classes

Chaque objet que vous manipulez dans le scénario ou par le code est régi par une classe. Une classe regroupe un ensemble de lignes de code, d'instructions, de fonctions, qui permettent de définir le comportement d'un objet lorsqu'il est déployé. Ces classes sont, soit intégrées dans le moteur de l'application Flash, soit inexistantes et c'est à vous de les ajouter. Certaines classes, intégrées à l'application Flash, sont transparentes et compilées par Flash à la publication sans même que l'on s'en aperçoive. C'est le cas des symboles de type bouton, `MovieClip`, des symboles graphiques, des champs de texte ou des composants vidéo. D'autres classes doivent être importées manuellement pour être utilisées, en les mentionnant en amorce du code par une commande du type : `import flash.events.Event;` Normalement, toutes les classes utilisées doivent être importées ainsi avant de s'y référer à travers, par exemple, un gestionnaire d'événements tel que celui-ci : `this.addEventListener(Event.ENTER_FRAME,defilementPanoramique);`. Mais, pour faciliter le codage, si nous développons directement dans l'interface auteur de Flash, la plupart des classes ne sont pas requises car elles sont automatiquement intégrées par l'API du logiciel. Ainsi, Il n'est pas nécessaire de mentionner l'importation de la classe `Event` pour l'utiliser dans notre gestionnaire d'événements.

Les sept lignes de code suivantes, placées à l'intérieur du bloc d'instruction de la fonction, font référence aux objets placés sur la scène principale.

```
////////// x
ciel_mc.x-=vitesseDefilement;
dune_mc.x-=vitesseDefilement*0.2;
arbre1_mc.x-=vitesseDefilement*0.3;
arbre2_mc.x+=vitesseDefilement*0.5;
arbre3_mc.x+=vitesseDefilement*0.8;
arbre4_mc.x+=vitesseDefilement*0.9;
```

Dans le contexte d'une fonction exécutée perpétuellement, ces actions redéfinissent la position en `x` de chacun des objets, un à un et en continu, de sorte que leur position s'incrémente automatiquement d'une certaine valeur. Cette valeur est définie en fonction de la variable

d'accélération `vitesseDefilement` initialisée plus haut. Certains objets observent une incrémentation positive (`+=`) et d'autres, une incrémentation négative (`-=`). Cette différenciation permet d'inverser le sens de défilement et d'accentuer l'effet de panoramique sur des objets distribués au premier plan, au regard de ceux placés à l'arrière-plan.

Nous déterminons ainsi, de façon arbitraire, le point de vue d'origine du spectateur. Selon la position de l'objet dans la scène et selon son éloignement par rapport à l'arrière-plan, nous indiquons également un coefficient (`*0.2` par exemple) qui permet de réduire son accélération de sorte que chaque objet se déplace à une vitesse différente. Cet effet de réduction du pas d'accélération permet d'accentuer plus encore l'effet de volume dans le décor, un peu à la manière d'une animation japonaise de type Manga où les plans qui se superposent sont souvent marqués. Pour renforcer encore l'effet volumétrique, un filtre flou est appliqué sur chaque objet, directement dans le scénario, avec des valeurs distinctes.

Le langage

Les propriétés en ActionScript 3

- `x`. `nomDuClip_mc.x` désigne l'abscisse `x`, soit la position horizontale, d'une occurrence de symbole clip ou `MovieClip` nommée `nomDuClip_mc`.
- `y`. `nomDuClip_mc.y` désigne l'ordonnée `y`, soit la position verticale, d'une occurrence de symbole clip nommée `nomDuClip_mc`.
- `width` et `height`. `nomDuClip_mc.width` et `nomDuClip_mc.height` désignent respectivement la largeur et la hauteur du même `MovieClip`.
- `rotation`. `nomDuClip_mc.rotation` désigne le degré de rotation de l'objet, défini sur une valeur comprise entre 0 et 360.
- `alpha`. `nomDuClip_mc.alpha` désigne l'opacité de l'objet. Cette valeur est comprise entre 0 et 1. 0 équivaut à 0 % d'opacité, donc, à de la transparence. 1 équivaut à 100 % d'opacité. Les valeurs intermédiaires sont exprimées en décimales. Ainsi, `nomDuClip_mc.alpha=0.4` désigne une valeur d'opacité de 40 %.
- `visible`. `nomDuClip.visible` permet de modifier la visibilité d'un objet dans la scène. Cette propriété se définit avec une valeur booléenne de type `true` ou `false`. Il n'existe pas de valeur intermédiaire. Cette propriété est moins gourmande en ressources graphiques que la propriété `alpha` et est recommandée pour masquer intégralement un objet, y compris la zone réactive qu'il possède éventuellement. La propriété `alpha`, effectivement, rend un objet invisible, mais conserve l'interactivité qui y est éventuellement attachée avec ActionScript. Par exemple, `nomDuClip.visible=false` rend l'objet `nomDuClip` invisible et non cliquable.
- `scaleX` et `scaleY`. `nomDuClip_mc.scaleX` et `nomDuClip_mc.scaleY` désignent respectivement l'échelle horizontale et verticale du même `MovieClip`. Cette valeur s'exprime également de 0 à 1, mais peut dépasser 1 si l'échelle doit être supérieure à 100 %. La valeur peut également être inférieure à 0 si l'on souhaite obtenir un effet miroir ou une symétrie sur un axe vertical (`scaleX`) ou horizontal (`scaleY`).

Ensuite, nous intervenons sur une autre propriété, la propriété `y`. Elle est utilisée ici pour redéfinir la position verticale du ciel pendant la progression du panoramique. En incrémentant

la valeur d'un entier positif, le ciel monte au début du panoramique et redescend dès que celui-ci atteint la limite du document :

```
////////// y
ciel_mc.y-=1;
```

De la même manière, la largeur et la hauteur du ciel sont modifiées avec les propriétés `width` et `height` afin que celui-ci donne l'illusion de se mouvoir. Les largeur et hauteur d'un symbole étant définies en pixels, la valeur est ici multipliée par 10 afin de la rendre perceptible, car un pas d'incrément de 1 pixel ne serait pas suffisamment visible :

```
////////// width et height
ciel_mc.width+=10;
ciel_mc.height+=10;
```

Comme pour les autres propriétés, `rotation` permet de faire pivoter l'objet avec un pas d'incrément défini en degrés. Pour atténuer l'effet et éviter que le ciel tourne complètement sur lui-même, la valeur est divisée par 10 :

```
////////// rotation
ciel_mc.rotation-=0.1;
```

L'animation d'une transparence se fait avec la propriété `alpha` définie sur une échelle de 0 à 1. Pour amortir l'effet, nous utilisons une valeur décimale faible. Cela équivaut aussi à diviser la valeur par un chiffre élevé :

```
////////// alpha
ciel_mc.alpha-=0.01;
dune_mc.alpha-=0.01;
arbre1_mc.alpha-=0.025;
arbre2_mc.alpha-=0.03;
arbre3_mc.alpha-=0.035;
arbre4_mc.alpha-=0.04;
```

Enfin, la déformation en x et en y se définit avec les propriétés `scaleX` et `scaleY`. Comme pour l'`alpha`, nous utilisons des valeurs comprises entre 0 et 1 pour déterminer une échelle de 0 % à 100 % par rapport à la taille originale du symbole. Pour éviter une déformation trop importante de la dune, nous divisons la valeur par 1 000 :

```
////////// scaleX et scaleY
dune_mc.scaleX+=0.001;
dune_mc.scaleY+=0.001;
```

Dans notre script, une action `trace` ponctue la fonction. Elle permet simplement de vérifier la position x du ciel que nous modifions et sert de référence pour l'ensemble de l'animation qui s'interrompt précisément par rapport à la position de cet objet :

```
// trace
trace(ciel_mc.x);
```

Enfin, la fonction s'achève avec une structure conditionnelle qui indique de stopper l'exécution de l'animation dès que la position x du ciel atteint une certaine valeur :

```
if (ciel_mc.x<=600) {
    this.removeEventListener(Event.ENTER_FRAME,defilementPanoramique);
}
```

Pour arrêter un gestionnaire d'événements, nous reprenons simplement le gestionnaire préalablement activé pour lequel nous substituons l'expression `addEventListener` à `removeEventListener` qui désigne clairement d'interrompre le gestionnaire. Sans cette action, en effet, l'animation continuerait, même si nous ne la percevons plus, jusqu'à des valeurs infinies, et risquerait d'alourdir considérablement l'exécution du programme. En interrompant les écouteurs lorsqu'ils ne sont plus nécessaires, vous optimisez vos animations.

À retenir

- Pour créer une animation perpétuelle, rythmée sur la cadence de la scène, nous utilisons un gestionnaire de type `Event.ENTER_FRAME`.
- Pour incrémenter ou diminuer une valeur, nous précédons le signe égal du signe plus (+) ou moins (-), selon que l'on souhaite augmenter la valeur ou la diminuer.
- Pour optimiser l'espace mémoire, en programmation, nous typons les valeurs et les objets avant de les utiliser dans les actions.
- La syntaxe chameau (en anglais : *Camel*) permet d'identifier rapidement le nom des fonctions et des occurrences d'objets créés, sur la scène ou dynamiquement.
- Les extensions `_mc`, `_btn`, `_txt` et `_video` permettent de véhiculer, dans le nom d'occurrence des objets, leur type et ainsi facilite leur identification dans le code.
- Les propriétés les plus couramment utilisées sont `x`, `y`, `width`, `height`, `rotation`, `alpha`, `visible` et `scaleX`, `scaleY`.
- Pour optimiser une animation, nous interrompons les écouteurs lorsqu'ils ne sont plus nécessaires, avec `removeEventListener`.

Gérer les propriétés avec des conditions

L'exemple utilisé dans cette section étend celui développé dans la section précédente. Il utilise en plus une structure conditionnelle et des variables d'ajustement qui permettent de déterminer automatiquement le sens de défilement d'un panoramique et sa vitesse, selon les valeurs attribuées à quelques variables. Dans cette section, nous allons voir comment centraliser le calcul de l'animation de l'ensemble des propriétés sur ces variables, définies en amont du programme.

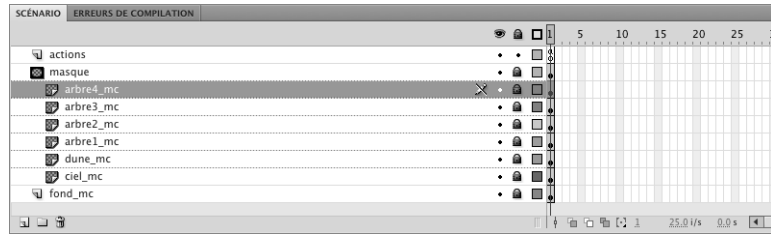


Exemples > `ch1_animationEnActionScript_2 fla`

Le document que nous utilisons présente la structure suivante : dans le scénario de la scène principale, au-dessus du calque `fond_mc`, clairement répartis vers des calques distincts et masqués, nous pouvons accéder, de même, à différents symboles de type `MovieClip` ayant tous un centre placé en haut et à gauche. Chacun de ces symboles dispose d'un nom d'occurrence qui le distingue des autres et lui permet d'être contrôlé par `ActionScript`. Pour faciliter l'identification de l'objet depuis la fenêtre `Actions`, le nom d'occurrence a été répété sur le calque qui contient chaque occurrence de symbole (voir Figure 1.3).

Figure 1.3

Structure du document.



En haut du scénario, un calque nommé actions contient le code suivant :

```
//----- initialisation
var sens:Number=1;
var vitesseDefilement:Number=5;
var limiteGauche:Number=(stage.stageWidth-vitesseDefilement)*-sens;
var limiteDroite:Number=vitesseDefilement*sens;

//----- actions
this.addEventListener(Event.ENTER_FRAME,defilementPanoramique);

function defilementPanoramique (evt:Event) {
    //////////// x
    ciel_mc.x-=(vitesseDefilement*1)*sens;
    dune_mc.x-=(vitesseDefilement*0.2)*sens;
    arbre1_mc.x-=(vitesseDefilement*0.3)*sens;
    arbre2_mc.x+=(vitesseDefilement*0.5)*sens;
    arbre3_mc.x+=(vitesseDefilement*0.8)*sens;
    arbre4_mc.x+=(vitesseDefilement*0.9)*sens;

    if (ciel_mc.x>limiteDroite) {
        sens=1;
    }
    if (ciel_mc.x<limiteGauche) {
        sens=-1;
    }

    //////////// y
    ciel_mc.y-=sens;

    //////////// width et height
    ciel_mc.width+=sens*10;
    ciel_mc.height+=sens*10;

    //////////// rotation
    ciel_mc.rotation-=sens/10;

    //////////// alpha
    arbre1_mc.alpha-=sens*0.025;
    arbre2_mc.alpha-=sens*0.03;
    arbre3_mc.alpha-=sens*0.035;
    arbre4_mc.alpha-=sens*0.04;

    //////////// scaleX et scaleY
    dune_mc.scaleX+=sens/1000;
    dune_mc.scaleY+=sens/1000;

    // trace
    trace(ciel_mc.x);
}
```


Ce code est structuré en deux parties. Une partie énumère les valeurs à configurer avant de lancer les actions. L'autre partie affiche les programmes à exécuter à partir de ces valeurs préalablement initialisées. Comme dans la section précédente, le programme créé pour le défilement du panoramique utilise d'abord la variable `vitesseDefilement` pour déterminer la position des objets de la scène, puis modifie leur direction selon la position courante de l'objet `ciel_mc`.

Tout d'abord, nous déclarons les valeurs à utiliser :

```
var sens:Number=1;
var vitesseDefilement:Number=5;
var limiteGauche:Number=(stage.stageWidth-vitesseDefilement)*-sens;
var limiteDroite:Number=(vitesseDefilement*-sens;
```

Avant de détailler le mode de calcul qui détermine ces valeurs, nous devons comprendre ce qui succède à la définition de ces variables. Dans la partie actions, située plus loin, vous relevez que les propriétés des objets du décor sont modifiées à partir des valeurs que nous définissons ici.

Revenons dans la partie initialisation. La variable `sens` (`var sens:Number=1`) déclare un nombre qui donne le sens de défilement du panoramique. Une valeur positive fait défiler le contenu de la scène d'abord vers la gauche, puis vers la droite et reprend ce mouvement perpétuellement. À l'inverse, une valeur négative le fait défiler d'abord vers la droite, puis vers la gauche, et réitère le mouvement perpétuellement.

Dans la ligne suivante, `vitesseDefilement` représente le coefficient d'accélération du défilement panoramique. Plus cette valeur est élevée, plus le défilement sera accéléré.

La variable `limiteGauche` déclare, elle, une autre valeur de type nombre. Cette variable définit la limite de défilement du décor à atteindre avant que la condition n'inverse le sens du défilement. Cette valeur résulte du calcul suivant : largeur de la scène (`stage.stageWidth`) moins le pas d'accélération (`vitesseDefilement`). Cette combinaison permet en réalité d'éviter de percevoir une rupture dans le décor lorsque celui-ci apparaît intégralement hors champ. En effet, considérons par exemple que le script exécute le défilement jusqu'à -800. Nous devons alors écrire ceci :

```
var limiteGauche:Number=stage.stageWidth*-sens
```

Comme ce script est exécuté après la vérification indiquée dans la structure conditionnelle, il aurait alors le temps de s'exécuter encore une dernière fois avant d'être réinitialisé seulement à la lecture suivante du script. Cela porterait le ciel à une position `x` de -805 et laisserait apparaître un blanc de 5 pixels de large, au-delà des dimensions limites du visuel. La valeur 5 correspondant à la valeur du pas d'incrément, nous la retranchons donc à la largeur limite du défilement pour prévenir du décalage induit par le fait que le calcul du repositionnement s'effectue seulement après la vérification de la condition. La scène qui mesure 800 pixels dans notre document, rapporte donc une largeur limite de 795 pixels.

À cela, la limite gauche du défilement panoramique, calculée à partir du positionnement du ciel, correspond en réalité au positionnement du ciel lorsqu'il atteint une valeur négative de -800. Il reste donc à inverser la valeur 795 obtenue par l'équation `stage.stageWidth-vitesse-Defilement`, en la multipliant par le sens du défilement afin d'obtenir une valeur négative, ou qui sinon corresponde à la valeur imposée par le sens voulu pour le défilement

de notre animation. Nous utilisons ainsi la valeur 1 pour un défilement vers la droite, ou -1 pour un défilement vers la gauche.

La variable `limiteDroite` déclare, elle, une valeur de type nombre également, mais qui détermine la limite du défilement à droite. Cette valeur est calculée sur le même principe que pour la limite gauche, excepté que nous nous basions sur le point d'origine de la scène qui vaut zéro, et non plus sur sa largeur. Le résultat obtenu est -5 ici, contre -795 pour la limite gauche. Le défilement oscille donc entre ces deux valeurs limites.

Puis, nous développons le programme :

```
//----- actions
this.addEventListener(Event.ENTER_FRAME,defilementPanoramique);
function defilementPanoramique (evt:Event) {
}
```

Les 7 lignes de code suivantes, placées à l'intérieur du bloc d'instruction de la fonction, font de nouveau référence aux objets placés sur la scène principale.

```
////////// x
ciel_mc.x+=(vitesseDefilement*1)*sens;
dune_mc.x+=(vitesseDefilement*0.2)*sens;
arbre1_mc.x+=(vitesseDefilement*0.3)*sens;
arbre2_mc.x+=(vitesseDefilement*0.5)*sens;
arbre3_mc.x+=(vitesseDefilement*0.8)*sens;
arbre4_mc.x+=(vitesseDefilement*0.9)*sens;
```

Les actions, ici, redéfinissent la position en x de chacun des objets, de sorte que leur position s'incrémente automatiquement d'une certaine valeur. Mais, cette valeur est désormais définie en fonction de deux coefficients : la variable d'accélération `vitesseDefilement` et la variable `sens`, initialisées plus haut.

Deux conditions déterminent ensuite à quel moment le lecteur Flash doit inverser la polarité du sens de défilement. La première condition indique de l'inverser lorsque le ciel atteint la limite droite du document, définie par la valeur `limiteDroite`. La seconde indique d'inverser également la valeur lorsque le ciel atteint la limite gauche avec la valeur `limiteGauche`. Ainsi, pour la première condition, nous pouvons : si la position x du ciel est supérieure à la valeur limite de droite qui vaut -5, alors, nous inversons le sens du pas d'incrémentation -1 en 1. Cette valeur affecte le calcul du positionnement des objets, en amont dans le code, de sorte que le défilement est également inversé, et inversement pour le défilement à gauche :

```
if (ciel_mc.x>limiteDroite) {
    sens=1;
}
if (ciel_mc.x<limiteGauche) {
    sens=-1;
}
```

Ensuite, nous intervenons sur les autres propriétés, en y associant les variables d'ajustement :

```
////////// y
ciel_mc.y-=sens;
////////// width et height
ciel_mc.width+=sens*10;
```

```

ciel_mc.height+=sens*10;
////////// rotation
ciel_mc.rotation-=sens/10;
////////// alpha
arbre1_mc.alpha-=sens*0.025;
arbre2_mc.alpha-=sens*0.03;
arbre3_mc.alpha-=sens*0.035;
arbre4_mc.alpha-=sens*0.04;
////////// scaleX et scaleY
dune_mc.scaleX+=sens/1000;
dune_mc.scaleY+=sens/1000;
// trace
trace(ciel_mc.x);

```

À retenir

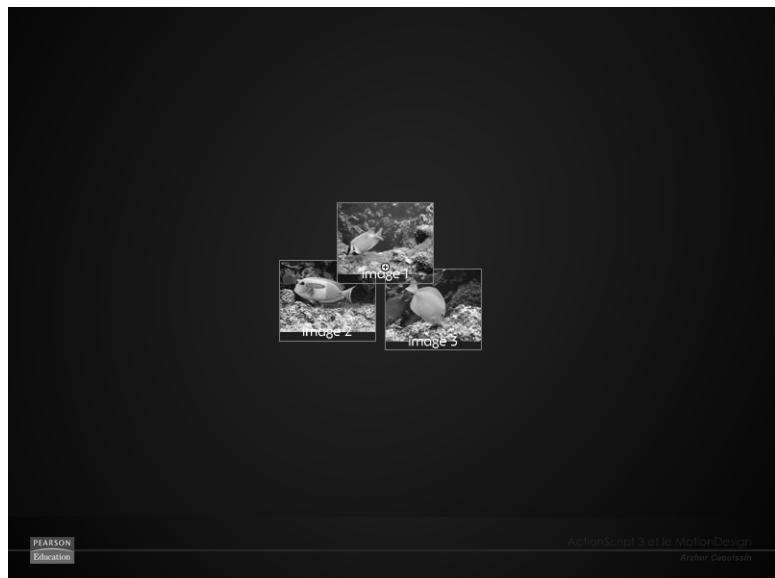
- Pour simplifier un programme, il est souhaitable de centraliser, sous la forme d'un dénominateur commun, ce qui est susceptible d'alléger le code, à travers une ou plusieurs variables par exemple.
- Les conditions permettent de modifier l'orientation d'une animation dès qu'elle atteint une certaine limite.

Gérer les accélérations

Dans cette section, des symboles sont disposés arbitrairement dans la scène. Lorsque le pointeur de la souris bouge, les symboles bougent aussi, mais avec une inertie (voir Figure 1.4). Nous allons apprendre ici à définir le mouvement d'un symbole en fonction de la position du pointeur, en lui affectant un effet de retard qui donne une illusion d'amortissement.

Figure 1.4

Aperçu
de l'exercice.



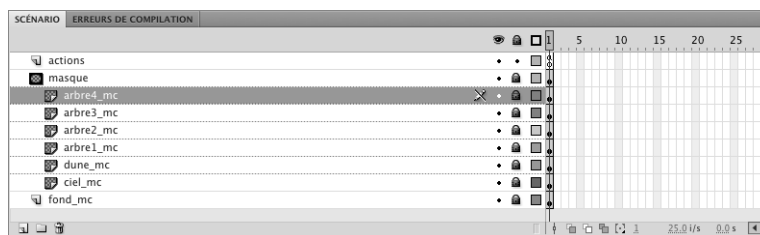


Exemples > chap1_animationEnActionScript_3.fl

Le document que nous utilisons présente la structure suivante : le décor masqué et situé au-dessus du calque `fond_mc`, est composé d'un dégradé et de trois symboles de type Movie-Clip, repartis chacun vers les calques de la scène principale. Chaque symbole est, en outre, placé par rapport à son centre, au milieu de la scène. Mais, à l'intérieur de chacun des symboles, les éléments sont repositionnés de sorte qu'ils ne se superposent pas complètement afin de permettre à l'utilisateur d'interagir avec ces objets, malgré leur position commune (voir Figure 1.5).

Figure 1.5

Structure du document.



Dans la partie supérieure, un calque `actions` présente le code utilisé pour l'animation.

L'astuce, ici, pour créer un effet de flottement des symboles sans toutefois redéfinir leur position relative les uns par rapport aux autres, est que nous utilisons un positionnement identique de tous les symboles. Mais chaque objet, dont le code contrôle le positionnement, se déplace avec un retardement. Si bien que nous avons l'illusion que les objets oscillent de part et d'autre en fonction de la position du pointeur, et reviennent toujours à leur position relative, prédéfinie dans la composition. Cette organisation structurelle permet de simplifier la gestion du déplacement du groupe que constitue l'ensemble des symboles sans avoir à multiplier inutilement les conteneurs et les lignes de code.

Du côté du code, toujours dans notre document, chaque clip est par ailleurs déplacé en incrémentant ses propriétés à travers un gestionnaire de type `ENTER_FRAME`, comme dans l'exemple précédent, mais des variables nous permettent de définir, en plus, des coefficients d'accélération pour générer l'effet de retardement.

En haut du scénario, le calque `actions` affiche le code suivant.

```
//----- initialisation
var retard:Number=20;

//----- actions
this.addEventListener(Event.ENTER_FRAME,navigation);

function navigation(evt:Event) {
    // x
    photo1_mc.x+=(mouseX-photo1_mc.x)/retard;
    photo2_mc.x+=(mouseX-photo2_mc.x)/(retard*2);
    photo3_mc.x+=(mouseX-photo3_mc.x)/(retard*3);
    // y
```

```

        photo1_mc.y+=(mouseY-photo1_mc.y)/retard;
        photo2_mc.y+=(mouseY-photo2_mc.y)/(retard*2);
        photo3_mc.y+=(mouseY-photo3_mc.y)/(retard*3);
    }

```

Dans un premier temps, nous déclarons les variables à utiliser :

```

//----- initialisation
var retard:Number=20;

```

La variable `retard` que nous créons permet de définir l'intensité de l'amortissement pour tous les symboles. Une valeur égale à 1 synchronise le mouvement de tous les symboles sur le pointeur. Une valeur élevée augmente le temps nécessaire à chaque symbole pour atteindre la position du pointeur et accentue, de ce fait, l'effet d'amortissement. La valeur 0 ne peut être appliquée car nous utilisons plus loin des divisions et l'on ne peut pas diviser par zéro.

Puis, nous développons le programme.

Dans la partie actions du codage, nous créons le gestionnaire qui appelle la fonction à exécuter. Le gestionnaire utilise la classe `Event` et l'événement `ENTER_FRAME` afin de permettre que le calcul du positionnement des symboles se fasse au rythme de la cadence de la scène. Pour une animation plus fluide, il suffit, par conséquent, d'augmenter la cadence de la scène qui est définie ici à 25 ips :

```

//----- actions
this.addEventListener(Event.ENTER_FRAME,navigation);

```

Plus loin, nous inscrivons la fonction qui rassemble les actions à exécuter :

```

function navigation(evt:Event) {
}

```

Le langage

Typage des nombres

Il est possible d'optimiser l'animation Flash de quelques fragments de millièmes de secondes en privilégiant un typage plus restrictif à un typage large. Par exemple, le typage `Number`, large, réserve l'espace mémoire requis pour stocker un chiffre entier ou décimal, positif ou négatif, codé sur 64 bits. Le typage `int`, lui, le restreint aux nombres entiers positifs et négatifs, à 32 bits. Le typage `int` est donc plus optimisé que le typage `Number` et est recommandé, dans la mesure où notre variable ne requiert pas une valeur décimale ou négative. Plus optimisé encore, le typage `uint` mentionne les nombres entiers et strictement positifs. Ainsi, nous pourrions tout aussi bien écrire : `var retard:int=20;` ou `var retard:uint=20;`. Cette recommandation est naturellement plus probante dans le cadre d'animations complexes et lourdes et se justifie un peu moins dans le cadre de cet exemple.

À l'intérieur du bloc d'instruction de la fonction, entre les deux accolades, nous plaçons les actions.

Le calcul que nous voulons effectuer repose sur un principe simple : il indique, pour chaque symbole, de le positionner en x et en y, à une valeur proportionnelle à la position du pointeur.

Pour bien comprendre le procédé, imaginons que l'on souhaite que chaque symbole soit positionné en suivant le pointeur en x et en y. Ce faisant, le groupe de symboles suivrait

strictement la position du pointeur si bien qu'il ne serait pas possible d'interagir avec les objets. Pour cela, nous indiquerions, pour chaque occurrence de symbole, ceci :

```
photo1_mc.x=mouseX ;
photo1_mc.y=mouseY ;
```

Afin que le déplacement dispose d'un retardement, la position de l'occurrence doit tendre progressivement vers la position du pointeur et non pas le suivre instantanément. Pour ce faire, nous incrémentons donc la position de l'occurrence d'une valeur qui diminue avec le temps.

Pour obtenir une valeur qui diminue avec le temps, nous faisons en sorte que l'occurrence conserve d'abord sa position initiale. Nous utilisons donc, dans notre calcul, `photo1_mc.x` qui détermine la position courante, en x, de l'objet que nous voulons rendre mobile : `photo1_mc`.

Nous veillons ensuite à ce que l'occurrence finisse par atteindre la position du pointeur. Nous utilisons aussi pour cela `mouseX`. Comme le calcul est répété indéfiniment, grâce à l'événement `ENTER_FRAME`, il est possible sans trop de complication de définir une valeur qui diminue progressivement avec le temps, comme suit.

```
photo1_mc.x+=(mouseX-photo1_mc.x)/retard;
photo1_mc.y+=(mouseY-photo1_mc.y)/retard;
```

Pour que la valeur diminue, nous incrémentons ici la position de l'occurrence, d'abord en x, de la différence comprise entre sa position d'origine et celle du pointeur. Puis, cette valeur est divisée par le coefficient de retardement pour obtenir une valeur de plus en plus faible. Comme l'action est réexécutée perpétuellement, à chaque réduction de la distance entre le point d'origine et la position du pointeur, il reste une distance de plus en plus faible à parcourir dans un même laps de temps. On obtient donc un effet de ralenti. Décliné en x et en y, pour les trois symboles présents sur la scène, nous obtenons :

```
photo1_mc.x+=(mouseX-photo1_mc.x)/retard;
photo2_mc.x+=(mouseX-photo2_mc.x)/retard;
photo3_mc.x+=(mouseX-photo3_mc.x)/retard;
photo1_mc.y+=(mouseY-photo1_mc.y)/retard;
photo2_mc.y+=(mouseY-photo2_mc.y)/retard;
photo3_mc.y+=(mouseY-photo3_mc.y)/retard;
```

Dans ce contexte, les occurrences se déplacent bien avec un amortissement par rapport à la position du pointeur, mais elles évoluent tous à la même vitesse, ce qui donne une impression d'immobilisme relatif. Pour que chaque occurrence avance désormais à un rythme différencié, il suffit de modifier l'impact du coefficient de retard en le divisant ou en le multipliant par un nombre indéterminé, ou en le substituant complètement à une valeur arbitraire tout simplement. Ce qui donne à présent :

```
//----- initialisation
var retard:Number=20;

//----- actions
this.addEventListener(Event.ENTER_FRAME,navigation);
function navigation(evt:Event) {
    // x
    photo1_mc.x+=(mouseX-photo1_mc.x)/retard;
    photo2_mc.x+=(mouseX-photo2_mc.x)/(retard*2);
    photo3_mc.x+=(mouseX-photo3_mc.x)/(retard*3);
```

```
// y
photo1_mc.y+=(mouseY-photo1_mc.y)/retard;
photo2_mc.y+=(mouseY-photo2_mc.y)/(retard*2);
photo3_mc.y+=(mouseY-photo3_mc.y)/(retard*3);
}
```

Le langage

Calculer avec la classe Math

Vous pouvez aussi déterminer des valeurs progressives en utilisant la classe Math, notamment pour obtenir une progression sinusoïdale. La classe Math offre différentes équations. Pour déterminer, par exemple, la valeur de position en abscisse d'un symbole photo_mc, inscrivez :

- `Photo_mc.x=Math.ceil(75.5)` ; arrondit 75.5 à l'entier positif supérieur le plus proche, soit 75.
- `Photo_mc.x=Math.floor(75.5)` ; arrondit 75.5 à l'entier positif inférieur le plus proche, soit 76.
- `Photo_mc.x=Math.pow(75.5,3)` ; élève la valeur passée en premier paramètre avec la seconde. On obtient ici 3 375.
- `Photo_mc.x=Math.random()` ; détermine une valeur aléatoire comprise entre 0 et 1, sous forme décimale au centième près. Pour obtenir un nombre entier compris entre 0 et 99, multipliez la valeur décimale obtenue par 100, voire arrondissez au nombre entier le plus proche, comme suit : `Photo_mc.x=Math.ceil(Math.random()*100)` ;.
- `Photo_mc.x=Math.sin(75.5)` ; renvoie le sinus de la valeur en paramètre. La valeur résultante est une valeur décimale comprise entre 1 et -1 exprimée en radians. Lorsque la valeur en paramètre progresse (comme i, dans le cadre d'une boucle avec `i++` par exemple), la valeur oscille de 1 à -1 de manière cyclique et permet de créer un mouvement d'oscillation (planète qui tourne, yoyo, pistons, etc). Le sinus de 0 vaut 1 et le cosinus de 0 vaut 0.
- `Photo_mc.x=Math.cos(75.5)` ; renvoie le cosinus de la valeur en paramètre. La valeur résultante est une valeur décimale comprise entre -1 et 1 selon le même principe que le sinus, mais la valeur est inversée. Le cosinus commence par -1.
- `Photo_mc.x=Math.sqrt(75.5)` ; renvoie la racine carrée de la valeur en paramètre. Ici, on obtient 8,65.

Conversion de degrés en radians et inversement

Les valeurs qui résultent de la classe Math.sinus ou Math.cosinus sont exprimées en radians. Un cercle compte 360 degrés ou 2 Pi radians. En ActionScript, on exprime la valeur de Pi avec la classe Math.PI. Ainsi, il est possible de convertir des degrés en radians, comme suit :

```
AngleEnRadians = AngleEnDegres × Math.PI/180
```

Ou des radians en degrés, comme suit :

```
AngleEnDegres = AngleEnRadians C 180/Math.PI
```

À retenir

- Pour apporter un effet d'amortissement à une animation gérée en ActionScript, on incrémente, à la position courante d'un objet, la position du pointeur moins la position de l'objet en question, que l'on divise par un coefficient. Plus le coefficient est élevé, plus l'accélération est amortie.

Gérer le défilement de panoramiques en boucle

Cet exemple propose un défilement panoramique ininterrompu. Il est programmé avec un gestionnaire de type `ENTER_FRAME` et contient des conditions. Ces conditions restaurent la position des éléments du panoramique lorsqu'il atteint une certaine limite. Les éléments composant le décor panoramique sont dupliqués de part et d'autre afin de permettre, lorsque le défilement atteint sa limite, d'effectuer son repositionnement de manière imperceptible (voir Figure 1.6).

Figure 1.6

Aperçu
de l'exercice.



Exemples > `ch1_animationEnActionScript_4 fla`

Le document que nous utilisons présente la structure suivante : au-dessus du calque `fond_mc` et sous le masque, le décor se compose de deux symboles, le ciel et une colline. Chacun de ces symboles contient une image préalablement travaillée dans Photoshop de sorte qu'elle peut être dupliquée plusieurs fois de suite horizontalement sans que l'on ne perçoive de rupture dans le graphisme. Nous obtenons une texture de type motif, qui raccorde aussi bien à droite qu'à gauche l'image référente située, elle, au centre de la composition. Ainsi, dans Flash, à l'intérieur de chaque symbole, l'image est répétée trois fois. Elles sont effectivement dupliquées de part et d'autre afin de permettre au symbole de sortir du champ visuel que constitue la largeur de la scène, sans pour autant laisser un vide dans le décor pendant le défilement. Ainsi, si le défilement s'effectue de gauche à droite, l'image placée à gauche comblera, en raccord, le trou que nous aurions pu percevoir si l'image n'avait pas été dupliquée de ce côté-ci. Inversement, si le défilement se fait de droite à gauche, l'image située à droite, avec raccord, permet de ne pas voir le trou qui résulte du déplacement du symbole, lorsque celui-ci est animé vers la gauche.

Chaque image est donc placée d'abord une fois à l'origine du symbole ($x = 0$ et $y = 0$). Puis, elle est dupliquée une fois à gauche et une fois à droite, toujours à l'intérieur du symbole qui contient l'image, de part et d'autre de l'emplacement initial de la première image. Les images répétées forment un ensemble continu et parfaitement homogène. La rupture entre chaque image est imperceptible. Les deux symboles, ainsi mis en forme, sont placés également à une abscisse de 0 sur la scène principale, en témoigne la position du centre de chacun de ces symboles (voir Figure 1.7).

Figure 1.7

Position du centre dans chaque symbole.

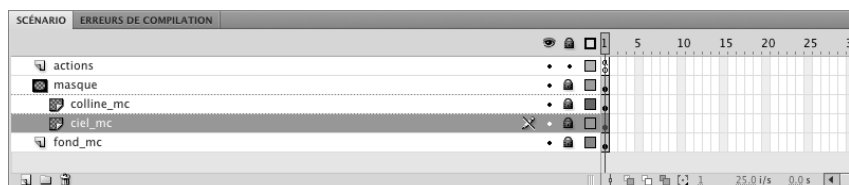


Réaliser une image de type motif, raccord, pour panoramiques. Pour réaliser une image que vous pourrez répéter de manière indéterminée de part et d'autre sans que l'on n'en perçoive la rupture, procédez comme suit :

1. Dans Photoshop, ouvrez une image qui représente un décor ou une texture.
2. Déverrouillez le calque principal et déplacez cette image hors champ, de sorte que l'un des côtés se retrouve au centre de la fenêtre de document. Le reste du document est à présent occupé par de la transparence.
3. Dupliquez l'image et placez la copie vers la zone vide en faisant se chevaucher légèrement les deux extrémités de chaque image au centre du document.
4. Puis, gomez (directement ou avec les masques de fusion) le bord de l'image dupliquée, pour révéler l'image originale située en arrière-plan. Fusionnez au besoin les deux calques. Vous devez éliminer toute rupture encore perceptible entre les deux images. Utilisez éventuellement l'outil Tampon pour retravailler plus précisément les zones de transition.
5. Enfin, recadrez l'image de sorte que chaque extrémité coïncide avec l'autre.
6. Exportez pour le Web au format JPEG ou PNG-24.

Figure 1.8

Structure du document.



En haut de la fenêtre de scénario (voir Figure 1.8), le calque actions affiche le code suivant :

```
//----- initialisation
var sensDefilement:Number=1;

//----- actions
this.addEventListener(Event.ENTER_FRAME,panoramiqueEnBoucle);

function panoramiqueEnBoucle (evt:Event) {
    // défilement
    ciel_mc.x+=3*sensDefilement;
    colline_mc.x+=6*sensDefilement;

    // boucle
    if (sensDefilement==1) {
        if (ciel_mc.x<=sensDefilement*(ciel_mc.width/3)) {
            ciel_mc.x=0;
        }
        if (colline_mc.x<=sensDefilement*(colline_mc.width/3)) {
            colline_mc.x=0;
        }
    } else {
        if (ciel_mc.x>=sensDefilement*(ciel_mc.width/3)) {
            ciel_mc.x=0;
        }
        if (colline_mc.x>=sensDefilement*(colline_mc.width/3)) {
            colline_mc.x=0;
        }
    }
}
```

Dans un premier temps, nous déclarons la variable à utiliser :

```
//----- initialisation
var sensDefilement:Number=1;
```

La variable `sensDefilement` indique ici le sens de défilement du panoramique. Une valeur de 1 induit un déplacement des symboles vers la droite. Une valeur de -1 indique un déplacement des symboles vers la gauche. Notez bien que le sens du déplacement agit sur le positionnement des objets dans la scène, et non sur la perception que l'on a de l'animation. Ainsi, si nous programmons le déplacement des symboles vers la droite, à la lecture de l'animation, nous percevons un mouvement panoramique qui s'oriente lui vers la gauche. Et, inversement, si les symboles se déplacent vers la gauche, nous percevons un panoramique qui évolue vers la droite.

Nous créons ensuite le gestionnaire d'événements qui permet d'exécuter le calcul du repositionnement en continu grâce à la classe `Event` et à l'événement `ENTER_FRAME`. Ce gestionnaire appelle la fonction que nous avons nommée `panoramiqueEnBoucle` :

```
//----- actions
this.addEventListener(Event.ENTER_FRAME,panoramiqueEnBoucle);
```

Nous plaçons alors, à la suite, la fonction en y introduisant, en paramètre, le rappel de la classe `Event` sans lequel notre fonction ne saurait s'exécuter :

```
function panoramiqueEnBoucle (evt:Event) {
}
```

À l'intérieur du bloc d'instruction de notre fonction, nous programmons d'abord le défilement des objets par incrémentation :

```
// défilement
ciel_mc.x+=3*sensDefilement;
colline_mc.x+=6*sensDefilement;
```

Nous pourrions utiliser une valeur arbitrairement positive ou négative directement ici, pour définir le sens du défilement et le pas, mais en la couplant avec notre variable, nous permettons d'automatiser l'inversion du défilement lorsque cela sera nécessaire. Il suffira alors de changer cette valeur une seule et unique fois, à l'initialisation, plus haut dans le code. Le sens du défilement, déclaré en amont, sera alors effectif pour l'ensemble de la fonction.

Le pas d'incrémenter n'ayant besoin d'être mentionné qu'une fois, nous le spécifions en revanche directement ici plutôt que de le stocker dans une autre variable. Nous utilisons un pas de 3 pixels pour le ciel afin qu'il progresse plus lentement que la colline pour laquelle nous avons un pas de 6 pixels. Cela permet d'accentuer ici aussi l'effet volumétrique ; effet que nous pouvons de nouveau combiner avec un léger filtre flou.

Une fois le défilement activé, nous pouvons définir les conditions à vérifier pour restaurer la position d'origine des symboles de sorte que l'effet boucle soit continu. Aussi, une première structure conditionnelle de type `if` doit être créée :

```
if (ciel_mc.x<=sensDefilement*(ciel_mc.width/3)) {
    // action
}
```

Cette condition vérifie que la position du ciel en `x` (`ciel_mc.x`) est inférieure ou égale à la position limite de l'image (`sensDefilement*(ciel_mc.width/3)`), juste avant qu'elle bascule hors champ, soit ici à `-800` pixels, car `800` pixels correspondent à la largeur effective de l'image et, en négatif, le déplacement se fait vers la gauche de la scène, donc, avant `0`.

Plus précisément, pour obtenir cette valeur, nous utilisons la variable `sensDefilement` qui indique le sens de défilement. Si le sens de défilement doit être positif (donc, défiler de gauche à droite), `sensDefilement` vaut `1`. Si le sens de défilement doit être négatif (donc, défiler de droite à gauche), `sensDefilement` vaut `-1`.

Au sens de défilement, nous ajoutons la largeur du symbole qui contient trois fois l'image de référence (`ciel_mc.width`), divisée par le nombre d'images qu'il contient, en l'occurrence `3` (parce que nous avons dupliqué l'image trois fois dans le symbole). Nous obtenons ainsi la largeur effective d'une et une seule image.

Une fois cette condition vérifiée à partir de l'ensemble de ces valeurs, nous replaçons l'objet à sa position d'origine qui est `0`. Nous obtenons ceci :

```
if (ciel_mc.x<=sensDefilement*(ciel_mc.width/3)) {
    ciel_mc.x=0;
}
```



Calculer un pas d'incrémenter. Lorsque vous animez les propriétés et les déplacements par incrémentation de valeur, et que vous souhaitez modifier le cours de l'animation dès qu'une certaine valeur est atteinte, pensez que la valeur du pas d'incrémenter est déterminante pour que la condition soit vérifiable. Par exemple, si je déplace un objet en `x`, sur un pas de `7` pixels, et que sa position

de départ est 0, je ne pourrai vérifier qu'une limite est atteinte que si je vérifie que sa nouvelle position est une valeur multiple de 7. Ainsi, lorsque je vérifie que l'objet atteint strictement 800 pixels avec : `if (monObjet_mc.x==800)`, avec un pas de 7, la condition ne sera jamais remplie, car 800 n'est pas un multiple de 7. Alors que la condition (`monObjet_mc.x>=800`), elle, sera exécutée, car 805, qui est un multiple de 7, est bien supérieur ou égal à 800. Afin d'atteindre une valeur multiple de 800, nous privilégions donc un pas de 5 pixels. Mais, même avec cette précaution, un trou de 5 pixels ($805 - 800 = 5$) peut faire apparaître encore une rupture dans le défilement. Il convient donc de bien calculer le rapport entre le pas d'incréméntation (5), la largeur du document (800) et la condition à remplir ($800-5$), pour que l'ensemble soit homogène et empêche le dernier pas d'incréméntation d'afficher un trou.

Éviter les images tremblantes. Vous obtenez toujours de meilleurs résultats à l'affichage lorsque les objets sont placés à des valeurs qui correspondent à des chiffres entiers. La position des objets est en effet déterminée en pixels et un pixel est par nature indivisible. Lorsqu'un objet atteint un positionnement exprimé en valeur décimale, vous risquez d'obtenir un scintillement léger, voire un flou, sur les images ou les textes que vous positionnez. Vérifiez donc toujours que vos symboles, dans une composition graphique, sont placés initialement à des valeurs entières. Pour cela, sélectionnez les objets individuellement et depuis l'Inspecteur de propriétés, ajustez si besoin les valeurs correspondant aux propriétés x et y.

La condition de bouclage est maintenant vérifiée pour le ciel. Nous déclinons cette vérification pour la colline. Nous obtenons :

```
if (ciel_mc.x<=sensDefilement*(ciel_mc.width/3)) {
    ciel_mc.x=0;
}
if (colline_mc.x<=sensDefilement*(colline_mc.width/3)) {
    colline_mc.x=0;
}
```

L'animation fonctionne, à ce stade, pour un défilement à gauche. Mais si nous voulons décliner la fonction pour un défilement à droite, nous devons inverser aussi le signe inférieur (<) présent dans la condition. Pour ce faire, nous doublons les actions en utilisant une nouvelle structure conditionnelle de type `if`, dans laquelle nous vérifions simplement la valeur du sens du défilement. Si la valeur est négative, alors, le signe est inférieur. Sinon, le signe devient supérieur. Nous obtenons :

```
//----- initialisation
var sensDefilement:Number=1;

//----- actions
this.addEventListener(Event.ENTER_FRAME,panoramiqueEnBoucle);
function panoramiqueEnBoucle (evt:Event) {
    // défilement
    ciel_mc.x+=3*sensDefilement;
    colline_mc.x+=6*sensDefilement;

    // boucle
    if (sensDefilement==1) {
        if (ciel_mc.x<=sensDefilement*(ciel_mc.width/3)) {
            ciel_mc.x=0;
        }
        if (colline_mc.x<=sensDefilement*(colline_mc.width/3)) {
            colline_mc.x=0;
        }
    }
}
```

```
    } else {  
        if (ciel_mc.x>=sensDefilement*(ciel_mc.width/3)) {  
            ciel_mc.x=0;  
        }  
        if (colline_mc.x>=sensDefilement*(colline_mc.width/3)) {  
            colline_mc.x=0;  
        }  
    }  
}
```

À retenir

- Pour créer une animation en boucle, il faut vérifier, avec une structure conditionnelle, les limites d'incrémentations des valeurs affectées aux propriétés des objets animés, et restaurer ensuite ces propriétés à leurs valeurs initiales, une fois les limites atteintes.
- Pour créer plus spécifiquement un panoramique en boucle, nous utilisons des images dont les extrémités se confondent afin de rendre imperceptible leur duplication dans l'environnement autour de Flash.
- Pour que l'animation en boucle soit homogène, nous devons vérifier que le pas d'incrémentations est compatible avec les valeurs vérifiées dans les conditions et avec les mesures du document.
- Une animation est plus fluide si les objets qu'elle contient sont positionnés sur des valeurs entières.
- Le typage uint ou int est plus optimisé que le typage Number, mais n'autorise que des nombres entiers, voire entiers positifs.
- L'utilisation de variables permet d'éviter de redéfinir manuellement une série de valeurs lorsqu'une seule information doit être modifiée, comme le sens de défilement d'un panoramique, par exemple.

Synthèse

Dans ce premier chapitre, vous avez vu comment réaliser des animations entièrement conçues en ActionScript, sans utiliser les options d'interpolation du scénario. Vous avez appris à les adapter à vos créations quelles qu'en soient les dimensions, à automatiser, grâce à la classe Event, certains processus en exploitant des variables, à générer des effets d'amortissement et des effets de boucle avec à l'utilisation de structures conditionnelles. Vous êtes en mesure, à présent, d'animer vos mises en forme en gardant les objets distribués sur la scène, sur une seule image du scénario, et simplifier ainsi vos futurs développements.

2

Interpolations et interactivité avec les classes *Tween* et *TweenMax*

Introduction

Nous avons vu, au chapitre précédent, comment animer des objets directement avec ActionScript. Pour réaliser des animations plus complexes que celles réalisées manuellement en ActionScript, et y ajouter une accélération, un effet de rebond, l'enchaînement de plusieurs interpolations successives ou y ajouter l'interpolation de propriétés, nous utilisons des classes ActionScript préprogrammées, plus simples à manipuler.

Le principe d'une classe est de rassembler sous la forme d'un fichier texte, un ensemble d'instructions à exécuter, pour un ou plusieurs paramètres donnés. Les classes d'animation que nous présentons dans ce chapitre permettent de spécifier en une seule instruction le nom d'un objet, les propriétés à animer, la durée de l'animation et le type d'effet d'accélération voulu ou non. C'est le compilateur Flash qui, à l'aide de la classe importée pendant la publication, convertit la ligne d'instruction en animation à la publication. Il devient donc enfantin, à partir de quelques lignes de code, de réaliser des interfaces riches, animées et dynamiques. Ces animations peuvent naturellement être ajustées selon la valeur des paramètres passés à l'exécution.

Ces classes se nomment Tween, TweenLite, TweenMax, Caurina, AS3AnimationSystem ou Twease. Si la classe Tween est native dans l'API de Flash, les classes TweenLite et TweenMax sont elles disponibles gratuitement sur le site <http://blog.greensock.com>. Les classes Caurina, AS3AnimationSystem et Twease, moins usitées, sont disponibles toutefois sur <http://code.google.com/p/tweener> sous la dénomination Tweener pour les deux premières, et sur <http://code.google.com/p/twease> pour la troisième. TweenMax permet de réaliser ce que propose TweenLite et représente la classe la plus fiable et aboutie de toutes. Nous abordons donc, dans ce chapitre, avec la classe Tween native déjà disponible dans votre application, et TweenMax, qui offre plus de possibilités et une plus grande stabilité.

Animer une galerie avec la classe *Tween* intégrée

Dans cet exemple, un décor multi-plans, composé de clips distincts, est animé dès l'exécution du document. Lorsque l'utilisateur survole l'une ou l'autre des flèches signalétiques situées de part et d'autre du décor panoramique, celui-ci se déplace dans un sens ou dans l'autre à des vitesses différentes. Des animations se succèdent les unes aux autres.

La spécificité de cet exemple est que chaque animation repose sur la classe Tween, native de Flash, simple d'utilisation (voir Figure 2.1).

Figure 2.1

Aperçu
de l'exercice.

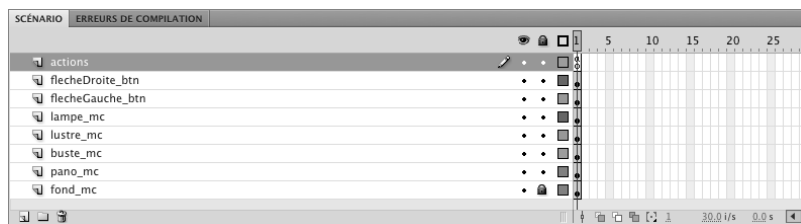


Exemples > ch2_interpolationEtInteractiviteAvecClassesTweenEtTweenMax_1 fla

Le document que nous utilisons présente la structure suivante : dans la scène, au-dessus du calque `fond_mc`, un ensemble de MovieClip et Bouton se répartissent sur divers calques. Chaque calque porte le nom de l'occurrence de l'objet qu'il contient (voir Figure 2.2). Le symbole `pano_mc` contient d'autres symboles dont certains sont cachés derrière un rideau. Ces symboles cachés sont également des MovieClip. Ils possèdent, dans leurs propres scénarios, des animations qui seront activées lors de leur survol par la souris (voir Figures 2.3 et 2.4).

Figure 2.2

Scénario
principal.

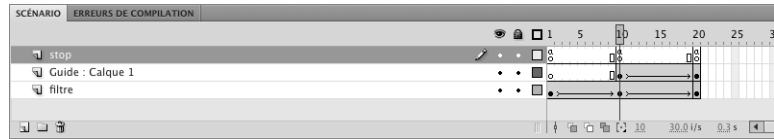
**Figure 2.3**

Aperçu des
symboles cachés
derrière le rideau.



Figure 2.4

Aperçu du scénario d'un des symboles cachés.



Dans la partie supérieure du scénario de la scène principale, le calque actions affiche le code suivant :

```
//----- initialisation
import fl.transitions.Tween;
import fl.transitions.easing.*;
import fl.transitions.TweenEvent;

var tweenMurX:Tween;
var tweenBusteX:Tween;
var tweenLustreX:Tween;
var tweenLampeX:Tween;
var tweenRideauX:Tween;

//----- actions
// animation d'introduction
tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, 715, 400, 4, true);
tweenBusteX= new Tween(buste_btn, "x", Strong.easeOut, 260, -260, 4, true);
tweenLustreX= new Tween(lustre_mc, "x", Strong.easeOut, 880, 400, 4, true);
tweenLampeX= new Tween(lampe_mc, "x", Strong.easeOut, 1400, 180, 4, true);

// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
    flecheGauche_btn.useHandCursor = false;
    tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, pano_mc.x, 690, 3, true);
    tweenBusteX= new Tween(buste_btn, "x", Strong.easeOut, buste_btn.x, 40, 3, true);
    tweenLustreX= new Tween(lustre_mc, "x", Strong.easeOut, lustre_mc.x, 700, 3, true);
    tweenLampeX= new Tween(lampe_mc, "x", Strong.easeOut, lampe_mc.x, 430, 3, true);
    tweenMurX.addEventListener(TweenEvent.MOTION_FINISH, apresTween1);
}
function apresTween1 (evt:TweenEvent) {
    flecheGauche_btn.visible=false;
    flecheDroite_btn.visible=true;
    tweenRideauX= new Tween(pano_mc.rideau_mc, "scaleX", Back.easeOut,
    pano_mc.rideau_mc.scaleX, 0.2, 2, true);
}

// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    flecheDroite_btn.useHandCursor = false;
    flecheGauche_btn.visible=true;
    flecheDroite_btn.visible=false;
    tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, pano_mc.x, 110, 4, true);
    tweenBusteX= new Tween(buste_btn, "x", Strong.easeOut, buste_btn.x, -560, 4, true);
    tweenLustreX= new Tween(lustre_mc, "x", Strong.easeOut, lustre_mc.x, 200, 4, true);
    tweenLampeX= new Tween(lampe_mc, "x", Strong.easeOut, lampe_mc.x, 100, 4, true);
}
```



```
//-----actions boutons cachés derrière le rideau, dans le clip pano_mc
// poupée
pano_mc.poupee_btn.addEventListener(MouseEvent.CLICK,roll);
pano_mc.poupee_btn.addEventListener(MouseEvent.CLICK,out);
// sac
pano_mc.sac_btn.addEventListener(MouseEvent.CLICK,roll);
pano_mc.sac_btn.addEventListener(MouseEvent.CLICK,out);

// robe
pano_mc.robe_btn.addEventListener(MouseEvent.CLICK,roll);
pano_mc.robe_btn.addEventListener(MouseEvent.CLICK,out);

// fonctions
function roll (evt:MouseEvent) {
    evt.currentTarget.gotoAndPlay(2);
}

function out (evt:MouseEvent) {
    evt.currentTarget.gotoAndPlay(11);
}
```

La programmation de notre animation se décompose en plusieurs étapes. Nous commençons par programmer le déplacement des symboles de la scène principale. Puis, nous ajoutons de l'interactivité sur les flèches droite et gauche placées aux bords du document. Enfin, nous ajoutons des actions pour contrôler l'interactivité avec les boutons cachés derrière le rideau de sorte qu'ils réagissent au passage du pointeur.

Le code se décompose comme suit. D'abord, nous importons les classes requises :

```
//----- initialisation
import fl.transitions.Tween;
import fl.transitions.easing.*;
import fl.transitions.TweenEvent;
```

Pour créer des animations à partir de la classe Tween, nous devons d'abord importer la classe Tween et ses classes dérivées. Nous pouvons alors déclarer de nouvelles animations et les associer à des événements souris.

Dans le code, le terme `import` indique de charger une classe externe non incluse dans l'API de Flash. En l'occurrence, la classe Tween native figure, non pas dans l'API du logiciel, mais dans un dossier placé à côté de l'application Flash sur le système. Il n'est donc pas nécessaire de placer physiquement ces classes près de notre document .fla pour les importer ni de les télécharger pour en disposer. Il suffit de les appeler directement là où elles se trouvent près de votre application sans les déplacer.



Localiser les classes natives de Flash. Sur Macintosh, les classes natives sont disponibles dans le dossier nommé `fl` qui se trouve dans le répertoire de l'application : Applications > Adobe Flash CS4 > Common > Configuration > ActionScript 3.0 > Projects > Flash > src > fl...

Sous Windows, elles sont localisées dans le répertoire de l'application Flash, de la même manière, depuis Program files > Adobe > Flash CS4.

Le terme `fl`, lorsqu'il est indiqué à la suite de `import`, indique à l'API de Flash de localiser ce répertoire qu'il sait identifier dans le dossier de l'application Flash, et à partir duquel il va chercher la classe ciblée dans le reste du chemin indiquée dans nos lignes de code. Dans notre exemple, Flash va donc importer les classes Tween et TweenEvent, qui se trouvent à la

racine du répertoire `transitions`, à l'intérieur de `fl`, puis, l'ensemble des classes disponibles contenues dans le dossier `easing`, lui-même situé à la racine du même dossier `transitions`.

Nous aurions pu simplifier en supprimant la dernière ligne et en remplaçant le terme `Tween` par un astérisque comme nous l'avons fait pour `easing`. Nous aurions alors obtenu :

```
import fl.transitions.*;
import fl.transitions.easing.*;
```

Toutes les classes alors contenues dans le répertoire mentionné avant l'astérisque sont systématiquement importées, à l'exception des classes qui figurent dans des sous-répertoires. C'est la raison pour laquelle nous garderions néanmoins la deuxième ligne. Mais en gardant perceptible le nom des classes principales que nous allons utiliser, nous conservons une visibilité sur le nom des classes qui spécifiquement nous intéressent ici et nous aident à mieux appréhender les instructions.

Nous abordons aussi, à la section suivante, le moyen de centraliser les classes ajoutées en redéfinissant les chemins de classe (`classPath`) à partir des préférences du système.



Que risque-t-on à abuser de l'astérisque à l'importation des classes ? Rien. Même si vous citez toutes les classes d'un répertoire avec un astérisque, seules celles qui sont vraiment utilisées dans le code de votre document Flash seront compilées à la publication. Cela n'alourdit donc nullement votre document.

Une fois que ces classes sont importées, les animations Tween peuvent être créées :

```
var tweenMurX:Tween;
var tweenBusteX:Tween;
var tweenLustreX:Tween;
var tweenLampeX:Tween;
var tweenRideauX:Tween;
```

D'abord, il faut réserver de nouveaux espaces mémoire pour contenir ces animations. Nous créons donc autant de variables que nous souhaitons créer d'interpolations (*tween* signifie interpolation en anglais). Chaque nouvelle variable doit être typée pour un espace de type `Tween`. `Tween` est la classe que nous avons importée et qui permet à Flash de comprendre de quoi il retourne lorsque nous l'employons.

Pour faciliter l'identification des interpolations, dans le nom des variables, nous introduisons arbitrairement le nom de l'objet à animer ainsi que la propriété concernée.

Maintenant que le typage est réalisé, nous pouvons animer :

```
//----- actions
// animation d'introduction
tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, 715, 400, 4, true);
tweenBusteX= new Tween(buste_btn, "x", Strong.easeOut, 260, -260, 4, true);
tweenLustreX= new Tween(lustre_mc, "x", Strong.easeOut, 880, 400, 4, true);
tweenLampeX= new Tween(lampe_mc, "x", Strong.easeOut, 1400, 180, 4, true);
```

Les lignes de code qui suivent l'initialisation reprennent les noms des interpolations que nous avons typées et y définissent un nouvel objet interpolation. Ces interpolations sont

caractérisées par l'objet `new Tween()` et une série de paramètres qui autorisent le contrôle de l'animation. Ces paramètres sont :

```
new Tween( objet, "propriété", accélération, valeur de départ,
valeur d'arrivée, durée, type de durée).
```

- **Objet.** Nom de l'objet à interpoler. Par exemple : `pano_mc`.
- **Propriété.** Entre guillemets, indique le nom de la propriété à animer. Par exemple : `x`, `y`, `scaleX`, `scaleY`, `rotation`, `alpha`, `width` ou `height`.
- **Accélération.** La méthode d'accélération de transition utilisée, en rapport à la classe `easing`. Par exemple : `Back`, `Bounce`, `Elastic`, `None`, `Regular` ou `Strong` (voir Tableau 2.1). Ces valeurs sont associées à un comportement qui détermine si l'effet doit être appliqué au début de l'animation, à la fin ou les deux simultanément, avec les paramètres respectifs : `easeIn`, `easeOut` ou `easeInOut`.
- **Valeur de départ.** La valeur à appliquer à la propriété de l'objet animé au début de l'interpolation. Cette valeur doit correspondre à la valeur attendue par la propriété animée. Ainsi, pour une transition sur l'axe des `x`, on utilisera un nombre entier positif, nul ou négatif ou une expression qui nous permet de la déterminer. L'expression `pano_mc.x` désigne par exemple la position courante de l'objet `pano_mc` en `x`, ou toute autre expression, pourvu que le résultat soit un nombre de préférence entier et positif, nul ou négatif. Une rotation appellera une valeur exprimée en degrés positifs ou négatifs (de 0 à 360 par exemple). Un `alpha` appellera une valeur décimale comprise entre 0 et 1.
- **Valeur d'arrivée.** La valeur de la propriété animée en fin d'interpolation, sur le même principe que la valeur de départ.
- **Durée.** La durée de l'interpolation. Un nombre entier positif est demandé.
- **Type de durée.** Requiert une valeur booléenne qui indique si la valeur indiquée pour la durée doit être exprimée en nombre d'images ou en secondes. Inscrivez `false` pour une valeur à exprimer en images. Inscrivez `true` pour une valeur exprimée en secondes.

Tableau 2.1 : Définition des méthodes de transition `easing`.

Back	Back permet de créer un effet de rebond unique avec une accélération. L'objet part dans une direction donnée, dépasse légèrement le point d'arrivée et revient au point d'arrivée défini par la valeur d'arrivée.
Bounce	Bounce génère un effet de rebond multiple qui, à la différence de <code>Elastic</code> , ne dépasse pas la valeur définie au point d'arrivée de l'interpolation. L'effet diminue progressivement et est également proportionnel à la distance entre la valeur de départ et la valeur d'arrivée de l'interpolation.
Elastic	Elastic génère un effet de rebond multiple qui dépasse la valeur définie au point d'arrivée de l'interpolation. L'effet diminue progressivement et est proportionnel à la distance entre la valeur de départ et la valeur d'arrivée de l'interpolation.
None	None neutralise les effets.
Strong	Strong ajoute une accélération prédéfinie simple à l'interpolation.
Regular	Regular permet de créer une interpolation linéaire sans effet, très légèrement plus amortie que <code>None</code> . Mais ceci est presque imperceptible.
easeIn, easeOut et easeInOut	Ces paramètres, ajoutés aux précédents, déterminent, en plus, si l'effet d'accélération spécifié par <code>Back</code> , <code>Bounce</code> , <code>Elastic</code> , <code>None</code> , <code>Regular</code> ou <code>Strong</code> , doit être appliqué au début de l'animation (on ajoute alors <code>.easeIn</code>), à la fin de l'animation (on ajoute alors <code>.easeOut</code>) ou les deux (on ajoute dans ce cas <code>.easeInOut</code>).

À ce stade de la programmation, l'animation d'introduction fonctionne de manière autonome. Nous poursuivons avec l'interactivité associée aux flèches gauche et droite :

```
// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
    // actions
}
```

D'abord, un écouteur est attaché à la flèche gauche. Cet écouteur appelle la classe MouseEvent pour désigner un événement de type souris. Nous précisons, derrière la classe MouseEvent, qu'il s'agit plus spécifiquement du survol de la souris sur l'objet auquel est rattaché l'écouteur, donc, la flèche gauche. Au survol, nous appelons l'exécution de la fonction nommée roll développée ci-dessous.

Dans cette fonction, qui reprend le nom de la classe appelée avec evt:MouseEvent, nous commençons par rendre invisible la main qui apparaît au survol du bouton flèche afin de rendre l'effet plus élégant :

```
flecheGauche_btn.useHandCursor = false;
```

À la suite, nous activons de nouvelles interpolations, comme nous l'avons fait pour l'animation d'introduction :

```
// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    flecheDroite_btn.useHandCursor = false;
    flecheGauche_btn.visible=true;
    flecheDroite_btn.visible=false;
    tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, pano_mc.x, 110, 4, true);
    tweenBusteX= new Tween(buste_btn, "x", Strong.easeOut, buste_btn.x, -560, 4, true);
    tweenLustreX= new Tween(lustre_mc, "x", Strong.easeOut, lustre_mc.x, 200, 4, true);
    tweenLampeX= new Tween(lampe_mc, "x", Strong.easeOut, lampe_mc.x, 100, 4, true);
}
```

Étant donné que nous avons déjà typé les interpolations pour l'introduction et que les propriétés et les objets animés ne changent pas, nous pouvons directement créer de nouvelles interpolations sans avoir à redéfinir de nouvelles variables. Ce sont les mêmes interpolations. Elles sont simplement mises à jour avec de nouvelles valeurs placées en paramètre.

Dans la première interpolation, comme dans les suivantes, nous relevons cependant une nuance :

```
tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, pano_mc.x, 110, 4, true);
```

La valeur utilisée pour désigner le point de départ de l'interpolation est une expression : pano_mc.x.

Plus loin, à la fin de la fonction appelée pour la flèche gauche, un écouteur est attaché à l'un des objets interpolation que nous venons d'utiliser. Il permet, grâce à la classe TweenEvent que nous avons importée plus haut, d'exécuter une nouvelle fonction lorsque l'interpolation est terminée. Nous associons alors, à la classe TweenEvent, l'événement MOTION_FINISH

pour appeler la nouvelle fonction (consultez le Tableau 2.2 pour connaître les différents événements disponibles avec cette classe) :

```
tweenMurX.addEventListener(TweenEvent.MOTION_FINISH, apresTween1);
}
function apresTween1 (evt:TweenEvent) {
}
```



Stabiliser une interpolation Tween. Si vous appliquez une nouvelle interpolation Tween sur un objet déjà en cours d'interpolation Tween, si la durée de la nouvelle interpolation est inférieure ou égale à l'interpolation en cours, elle perd le contrôle de l'objet animé. En revanche, si cette interpolation est plus longue, elle devient prioritaire. Pour stabiliser une interpolation Tween qui ne s'achève pas correctement, il convient donc de modifier la durée de chaque interpolation de sorte que la dernière exécutée soit toujours plus longue que celle éventuellement en cours ou d'interrompre les autres interpolations (voir Tableau 2.2).

Tableau 2.2 : Définition des enchaînements de la classe TweenEvent.

MOTION_CHANGE	Indique que l'interpolation se joue.
MOTION_FINISH	Indique que l'interpolation est terminée.
MOTION_LOOP	Indique que la lecture de l'interpolation a repris depuis le début, en mode boucle.
MOTION_RESUME	Indique que la lecture de l'interpolation a repris, après une pause.
MOTION_START	Indique que la lecture du mouvement a commencé.
MOTION_STOP	Indique que l'objet Tween a été interrompu par un <code>.stop()</code> . Cet événement peut être déployé pour prévenir, par exemple, de toute intervention de l'utilisateur qui souhaite interrompre une interpolation en cours d'exécution afin d'éviter que l'interpolation ne bogue. Notez cependant que la classe Tween supporte mal l'interruption d'interpolations en cours et de ce fait, oblige à réaliser des animations rapides et de courte durée. Lorsqu'une interpolation se joue et qu'un utilisateur souhaite cliquer, par exemple, sur un objet en mouvement pour appeler un contenu, il est souhaitable d'interrompre avec <code>LeNomDeMonTween.stop()</code> l'animation en cours. Le cas échéant, le document exécuté peut devenir très instable. Pour des animations plus stables et plus longues, la classe TweenMax, qui supporte mieux cette contrainte, sera privilégiée.

Dans cette nouvelle fonction, nous demandons de modifier la visibilité des flèches puis de lancer une nouvelle interpolation qui va, elle, déformer le rideau de manière à révéler les boutons cachés à l'arrière-plan de celui-ci. Nous utilisons ici la propriété `scaleX` et des valeurs décimales car l'échelle est définie à 0 pour 0 %, et à 1 pour 100 % :

```
tweenMurX.addEventListener(TweenEvent.MOTION_FINISH, apresTween1);
}
function apresTween1 (evt:TweenEvent) {
    flecheGauche_btn.visible=false;
    flecheDroite_btn.visible=true;
    tweenRideauX= new Tween(pano_mc.rideau_mc, "scaleX", Back.easeOut,
        ➤ pano_mc.rideau_mc.scaleX, 0.2, 2, true);
}
```

Avant l'interpolation, ou après, peu importe, nous rendons également invisible le bouton flèche gauche pour éviter que l'utilisateur ne lance à plusieurs reprises l'animation. Comme l'animation déplace le décor latéralement, si nous l'exécutons indéfiniment, le décor finirait par sortir complètement de la scène. Comme nous utilisons ici un bouton pour amorcer

l'animation, il n'est pas nécessaire de recourir à des structures conditionnelles comme nous l'aurions fait au chapitre précédent. Il suffit, en effet, après l'interpolation de masquer intégralement ce qui permet de lancer l'animation, le bouton flèche gauche :

```
flecheGauche_btn.visible=false;
```

Inversement, si nous masquons la flèche gauche pour éviter de sortir vers la gauche, nous devons permettre à l'utilisateur de revenir vers la droite, de l'autre côté du décor, lorsque nous sommes à l'autre extrémité de celui-ci. Nous forçons donc la visibilité de la flèche droite au moment où la flèche gauche disparaît, afin de prévenir le cas où la fonction associée à la flèche droite (qui rend invisible la flèche droite) aura été exécutée préalablement par d'autres fonctions :

```
flecheDroite_btn.visible=true;
```

À la suite, de la même manière que nous avons créé l'action pour la flèche gauche, figurent celles utilisées pour la flèche droite, mais sans enchaînement d'animation car le rideau n'est présent que d'un côté de la scène :

```
// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    flecheDroite_btn.useHandCursor = false;
    flecheGauche_btn.visible=true;
    flecheDroite_btn.visible=false;
    tweenMurX= new Tween(pano_mc, "x", Strong.easeOut, pano_mc.x, 110, 4, true);
    tweenBusteX= new Tween(buste_btn, "x", Strong.easeOut, buste_btn.x, -560, 4, true);
    tweenLustreX= new Tween(lustre_mc, "x", Strong.easeOut, lustre_mc.x, 200, 4, true);
    tweenLampeX= new Tween(lampe_mc, "x", Strong.easeOut, lampe_mc.x, 100, 4, true);
}
```

Enfin, nous terminons avec des actions placées sur les boutons cachés, qui permettent de signaler une possible interaction, au passage de la souris. Les actions appelées à travers les écouteurs lancent l'animation contenue dans le scénario de chaque bouton.

Chaque écouteur dont nous disposons utilise la classe `MouseEvent` à laquelle chacune des deux fonctions `roll` et `out` se réfère *via* le terme `evt`. En reprenant, dans chaque fonction, le terme `evt`, nous pouvons alors invoquer la classe `MouseEvent`. En y associant en plus la propriété `currentTarget`, nous ciblons, plus spécifiquement, chacun des objets écoutés véhiculé par la classe `MouseEvent`. Ainsi, il n'est pas nécessaire de répéter maintes fois la même fonction et la décliner pour chaque bouton si les actions sont communes. Il suffit de remplacer le nom de l'occurrence ciblée (`pano_mc.poupee_btn`, `pano_mc.sac_btn` et `pano_mc.robe_btn`) par l'expression `evt.currentTarget` et les fonctions `roll` et `out` seront actives pour chacun des écouteurs s'y référant (voir Figure 2.4).

Cette animation s'arrête au premier stop rencontré. La fonction nommée `out` prolonge ces animations lorsque le pointeur quitte la surface des mêmes boutons.

```
// poupée
pano_mc.poupee_btn.addEventListener(MouseEvent.CLICK,roll);
pano_mc.poupee_btn.addEventListener(MouseEvent.CLICK,out);
// sac
pano_mc.sac_btn.addEventListener(MouseEvent.CLICK,roll);
pano_mc.sac_btn.addEventListener(MouseEvent.CLICK,out);
```

```
// robe
pano_mc.robe_btn.addEventListener(MouseEvent.CLICK,roll);
pano_mc.robe_btn.addEventListener(MouseEvent.CLICK,out);

// fonctions
function roll (evt:MouseEvent) {
    evt.currentTarget.gotoAndPlay(2);
}

function out (evt:MouseEvent) {
    evt.currentTarget.gotoAndPlay(11);
}
```

À retenir

- La classe Tween est simple d'utilisation car elle ne nécessite pas de disposer des classes correspondantes à côté du document Flash pour les utiliser. Il suffit de les importer en ciblant les répertoires situés dans le dossier Applications du système.
- Il est possible d'enchaîner plusieurs animations en utilisant la classe TweenEvent, suivie de l'événement MOTION_FINISH.
- Les interpolations de la classe Tween permettent d'animer les propriétés x, y, scaleX, scaleY, rotation, alpha, width et height d'objets affichés par le lecteur Flash.
- Les valeurs utilisées en paramètre pour définir les caractéristiques d'une interpolation Tween peuvent résulter d'expressions.
- Les classes Tween doivent cependant être utilisées avec parcimonie pour préserver la stabilité de vos créations. Pour des animations plus élaborées, préférez la classe TweenMax.
- La classe Tween supporte mal les interventions utilisateur qui interrompent son processus. Privilégiez des interpolations de courte durée pour palier à cette contrainte ou utilisez la classe TweenMax.
- La propriété currentTarget permet de cibler directement plusieurs occurrences à travers une seule fonction et évite de créer des fonctions similaires contenant les mêmes instructions.

Animer une carte interactive avec la classe *TweenMax*

La classe Tween native de Flash est simple, mais très vite limitée en possibilités dès lors que l'on souhaite véritablement réaliser des animations plus élaborées. Il est difficile en effet de multiplier les propriétés sur un même objet sans accroître également le nombre de lignes de code. La multiplication des lignes de code avec des interpolations de type Tween rend d'autant plus instable le fichier qui les exécute et de fait, limite réellement l'utilisabilité de la classe native.

Pour disposer de classes d'interpolation plus souples et plus stables, nous préférons utiliser la classe TweenMax disponible gratuitement sur le site <http://blog.greensock.com/tweenmaxas3>. Cette classe est également disponible en AS2 à l'adresse <http://blog.greensock.com/tweenmaxas2>. Elle permet, outre l'interpolation à partir de propriétés, de gérer l'accélération bien sûr, mais aussi de placer sur l'interpolation en cours une trajectoire de type Bézier pour définir précisément un mouvement. Elle permet également de gérer des effets de transition de couleur sur des objets.

Dans cette section, nous allons étudier le fonctionnement d'une carte géographique interactive en combinant, au sein de mêmes transitions, l'animation de plusieurs propriétés, de trajectoires définies avec une courbe de Bézier et l'enchaînement de ces transitions (voir Figure 2.5).

Figure 2.5

Aperçu
de l'exercice.

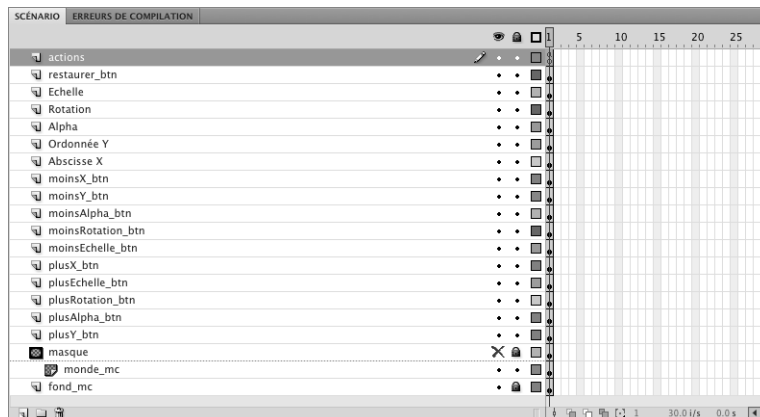


Exemples > ch2_interpolationEtInteractiviteAvecClassesTweenEtTweenMax_2 fla

Le document que nous utilisons présente la structure suivante : dans la scène, au-dessus du calque fond_mc, figure d'abord la carte du monde en partie masquée de sorte qu'en l'agrandissant, elle ne déborde pas sur le pied de page du document. Au-dessus, clairement identifiés et répartis vers les calques, se trouvent les boutons plus et moins affectés à la manipulation des propriétés et leurs légendes respectives (voir Figure 2.6).

Figure 2.6

Aperçu
de la fenêtre
scénario.



Dans le calque actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

restaurer_btn.visible=false;

var tweenMonde:TweenMax;

//----- actions

//// anim intro
TweenMax.from(monde_mc, 5, {scaleX:10, scaleY:10, rotation:90, x:700, y:1200,
➤ bezier:[{x:290, y:143}, {x:300, y:345}], delay:0.2, ease:Strong.easeInOut});

//// actions boutons

// Abscisse x
plusX_btn.addEventListener(MouseEvent.CLICK,plus1);
function plus1 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {x:monde_mc.x-100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
moinsX_btn.addEventListener(MouseEvent.CLICK,moins1);
function moins1 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {x:monde_mc.x+100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}

// Ordonnée y
plusY_btn.addEventListener(MouseEvent.CLICK,plus2);
function plus2 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {y:monde_mc.y-100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
moinsY_btn.addEventListener(MouseEvent.CLICK,moins2);
function moins2 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {y:monde_mc.y+100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}

// Rotation
plusRotation_btn.addEventListener(MouseEvent.CLICK,plus3);
function plus3 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {rotation:monde_mc.rotation-15, delay:0,
    ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
moinsRotation_btn.addEventListener(MouseEvent.CLICK,moins3);
function moins3 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {rotation:monde_mc.rotation+15, delay:0,
    ease:Strong.easeOut});
    restaurer_btn.visible=true;
}

// Alpha
plusAlpha_btn.addEventListener(MouseEvent.CLICK,plus4);
function plus4 (evt:MouseEvent) {
```

```

        if (monde_mc.alpha<1) {
            TweenMax.to(monde_mc, 1, {alpha:monde_mc.alpha+0.1, delay:0, ease:Strong.easeOut});
            restaurer_btn.visible=true;
        }
    }
    moinsAlpha_btn.addEventListener(MouseEvent.CLICK,moins4);
    function moins4 (evt:MouseEvent) {
        if (monde_mc.alpha>0) {
            TweenMax.to(monde_mc, 1, {alpha:monde_mc.alpha-0.1, delay:0, ease:Strong.easeOut});
            restaurer_btn.visible=true;
        }
    }

    // Echelle
    plusEchelle_btn.addEventListener(MouseEvent.CLICK,plus5);
    function plus5 (evt:MouseEvent) {
        TweenMax.to(monde_mc, 2, {scaleX:monde_mc.scaleX+0.5, scaleY:monde_mc.scaleY+0.5,
            delay:0, ease:Strong.easeOut});
        restaurer_btn.visible=true;
    }
    moinsEchelle_btn.addEventListener(MouseEvent.CLICK,moins5);
    function moins5 (evt:MouseEvent) {
        TweenMax.to(monde_mc, 2, {scaleX:monde_mc.scaleX-0.5, scaleY:monde_mc.scaleY-0.5,
            delay:0, ease:Strong.easeOut});
        restaurer_btn.visible=true;
    }

    // Restaurer
    restaurer_btn.addEventListener(MouseEvent.CLICK,restaurerCarte);
    function restaurerCarte (evt:MouseEvent) {
        tweenMonde= TweenMax.to(monde_mc, 5, {scaleX:1, scaleY:1, x:400, y:270, rotation:1,
            alpha:1, delay:0, ease:Strong.easeInOut});
        tweenMonde.addEventListener(TweenEvent.COMPLETE, tweenFini1);
    }
    function tweenFini1 (evt:TweenEvent) {
        restaurer_btn.visible=false;
    }
}

```

La classe TweenMax, à la différence de la classe native Tween, n'est pas présente dans le répertoire `fl` situé dans le répertoire de l'application Flash. Vous devez donc avoir téléchargé cette classe.

Une fois la classe téléchargée et dézippée, le dossier obtenu contient un dossier nommé `gs` qui rassemble l'ensemble des classes et sous-classes utiles pour gérer les interpolations de la classe TweenMax. Vous devez placer ce dossier à côté de votre document Flash, à l'intérieur de votre projet jusqu'à la publication des fichiers SWF.



Mise à jour de la classe GreenSock TweenMax. Les classes Greensock TweenLight et TweenMax étant régulièrement mises à jour, la structure et le nom des éléments qui les composent sont sujets à modification. Vous adapterez naturellement nos commentaires présents tout au long de l'ouvrage en fonction de ces éventuelles mises à jour. Le dossier de classes `gs` que nous utilisons dans cet ouvrage étant apposé aux exemples du livre, vous disposez déjà d'une structure valide pour l'ensemble des fichiers et ne nécessitez pas de la télécharger à nouveau, y compris pour tout développement d'interpolation de type TweenMax que vous seriez amené à produire vous-même par la suite.

Que faire des classes après compilation des SWF ?

Le code contenu dans les classes appelées avec la commande `import` en ActionScript, à l'intérieur de votre document, sera compilé directement dans le SWF. Il n'est donc pas nécessaire de placer les classes sur le serveur qui héberge votre site avec les fichiers SWF. Néanmoins, il est souhaitable de les conserver près de votre document source Flash (.fla) afin de permettre les éventuelles mises à jour et la recompilation de vos animations au format SWF.

Centraliser les classes ajoutées avec les chemins de classe (`classPath`)

Il est possible de centraliser les classes que vous utilisez dans un autre répertoire et de s'y référer pour chaque nouveau document pour lequel vous souhaitez les utiliser, ponctuellement pour le document ouvert ou définitivement pour l'ensemble des nouveaux documents Flash. La première méthode consiste à définir un chemin d'accès automatique pour les classes natives (`classPath`), chemin invoqué à la publication du document actif en redéfinissant ses paramètres de publication. La seconde méthode consiste à redéfinir ce même chemin de classes natives, directement dans les préférences de l'application Flash, de sorte que chaque nouveau document dispose des nouvelles classes.

Dans les deux cas, il devient alors possible d'appeler la classe ajoutée uniquement par son nom, et sans la recopier physiquement dans votre projet, comme nous le faisons déjà avec la classe `Tween` native.

Pour importer une classe pour le document actif uniquement, il faut redéfinir les paramètres de publication du document ouvert. Faites Fichier > Paramètres de publication (ou depuis l'Inspecteur de propriétés, cliquez sur le bouton du haut intitulé Modifier). Au sommet de la boîte de dialogue, dans l'onglet Flash, à droite de la liste de version du langage ActionScript, cliquez sur le bouton Paramètres. Dans la nouvelle boîte de dialogue, en bas, ajoutez un chemin en cliquant sur le bouton plus. Puis, sélectionnez la classe à ajouter en cliquant sur le bouton de sélection matérialisé par un petit dossier, à droite du bouton moins. Confirmez l'importation et validez toutes les fenêtres. La classe est intégrée au document.

Pour importer une classe pour l'ensemble des nouveaux documents, vous devrez redéfinir les préférences de Flash. Dans les préférences de l'application (Fichier > Préférences), cliquez d'abord sur la catégorie ActionScript, puis sur le bouton Paramètres d'ActionScript 3.0, situé en bas de la fenêtre. Dans la boîte de dialogue qui apparaît, dans la deuxième liste d'affichage nommée Dossiers, contenant des fichiers de classes ActionScript, cliquez sur le bouton de sélection de répertoire pour sélectionner le répertoire de classes et l'intégrer désormais à vos futurs documents.

À côté du dossier `gs` que vous avez téléchargé, figurent des documents au format SWF, respectivement nommés `PluginExplorer.swf` et `TweenLiteBasics.swf`. Ces fichiers vous assisteront sur l'édition des lignes de code pour générer une interpolation de type `TweenMax`. Ces assistants sont très utiles, notamment pour le calcul d'une trajectoire de mouvement de type Bézier que nous utilisons pour déplacer des objets (voir Figure 2.7).



L'assistant TweenMax. Pour utiliser l'assistant TweenMax, ouvrez, avec le lecteur Flash installé sur votre système, le fichier nommé `PluginExplorer.swf`. Puis, cliquez sur le bouton `Example` situé près du type d'interpolation de votre choix. Une nouvelle fenêtre apparaît et affiche les lignes de code relatives à la propriété choisie. Vous pouvez alors modifier les réglages dans la partie droite de la fenêtre pour ajuster le script ou déplacer directement les options de contrôle dans la zone d'aperçu située à gauche de la fenêtre (voir Figure 2.8). Pour contrôler une courbe de Bézier, par exemple, cliquez successivement sur le tracé qui figure dans l'aperçu à gauche, puis déplacez les points créés afin

d'ajuster instantanément le tracé. À droite, activez éventuellement l'option Tween through bezier points pour que le tracé passe par les points créés au lieu d'y être simplement aimanté. Cliquez ensuite sur le bouton TWEEN pour lancer un aperçu de l'animation obtenue. Appuyez sur Reset pour réinitialiser les paramètres.

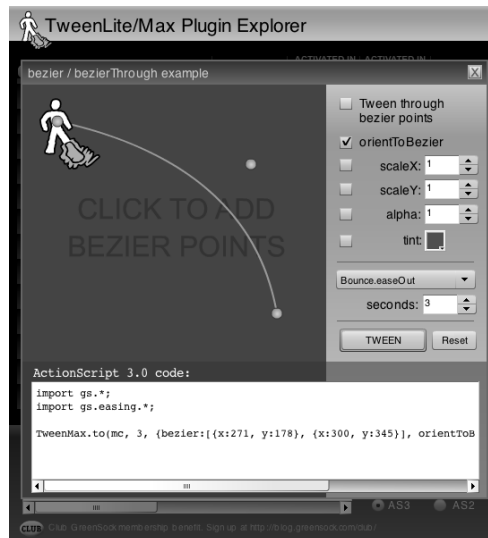
Figure 2.7

Aperçu de
l'assistant
TweenMax.



Figure 2.8

Détail de la
configuration
d'une propriété.



Pour récupérer le code généré par l'application, dans la zone de code située en bas de la fenêtre, copiez-le directement en appuyant sur Cmd+A, puis Cmd+C (Mac) ou Ctrl+A, puis Ctrl+C (Windows). Puis collez le code directement dans la fenêtre Actions de votre document Flash en faisant Cmd+V (Mac) ou Ctrl+V (Windows).

Dans le code, nous démarrons avec l'importation des classes requises.

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
```

Les classes importées sont respectivement celles contenues dans le répertoire `gs` pour les interpolations, dans le répertoire `easing` pour permettre les effets de transition et d'accélération, et dans le répertoire `events` pour autoriser l'enchaînement d'animations ou l'exécution de fonctions après les animations (événements).

Puis, nous masquons le bouton `restaurer_btn` qui sert à restaurer la position initiale de la carte, car elle n'est pas encore modifiée. C'est uniquement, lors d'une modification, que nous rendons ce bouton visible afin de permettre la réinitialisation de la carte à ses paramètres d'origine :

```
restaurer_btn.visible=false;
```

Les occurrences d'animation issues de la classe `TweenMax` ne requièrent pas d'être typées. Mais nous déclarons néanmoins ici une nouvelle occurrence d'animation afin de permettre d'y attacher, plus loin dans le code, un écouteur. Cela va permettre d'exécuter d'autres fonctions lorsque celle-ci sera terminée et introduire, si vous le souhaitez, d'autres interpolations, comme nous l'avons vu avec la classe `Tween` native dans la section précédente :

```
var tweenMonde:TweenMax;
```

Puis, viennent les actions :

```
//----- actions

//// anim intro
TweenMax.from(monde_mc, 5, {scaleX:10, scaleY:10, rotation:90, x:700, y:1200,
    ➤ bezier:[{x:290, y:143}, {x:300, y:345}], delay:0.2, ease:Strong.easeInOut});
```

D'abord, nous créons une première interpolation `TweenMax`. Celle-ci interpole les propriétés de l'objet `monde_mc` à partir des valeurs indiquées en paramètre, vers les valeurs courantes de l'objet sur la scène (grâce à l'instruction `from`). L'animation dure 5 secondes (5) et démarre un cinquième de seconde après l'exécution du Flash (`delay:0.2`). Sa transition est de type progressive (`Strong`) et la progression est marquée à la fois en entrée et en sortie (`easeInOut`). Les propriétés animées sont respectivement l'échelle (`scaleX` et `scaleY`), la rotation (`rotation`) et la position (`x` et `y`). Cette animation enfin suit une trajectoire de type Bézier matérialisée par deux points d'ancrage dont les coordonnées sont stockées dans un tableau (`bezier:[{x:290, y:143}, {x:300, y:345}]`).



Définir les coordonnées d'une courbe de Bézier pour la classe `TweenMax`. Pour utiliser le paramètre `bezier` dans une transition `TweenMax`, vous devez connaître au préalable les coordonnées des points d'inflexion de la courbe (sauf à utiliser l'assistant `TweenMax` présenté plus haut). Pour connaître les coordonnées des points d'inflexion, utilisez les repères de composition de votre document Flash. Dans le menu principal de Flash, choisissez `Affichage > Règles`. Puis, glisser-déposez des repères depuis la règle située à gauche, vers un point d'inflexion, pour en connaître l'abscisse `x`. De même, glisser-déposez un guide de la règle du haut, vers un point d'inflexion, pour en connaître l'ordonnée `y`.

Viennent ensuite les actions relatives à chaque bouton plus et moins, pour chaque propriété :

```
//// actions boutons

// Abscisse x
plusX_btn.addEventListener(MouseEvent.CLICK,plus1);
function plus1 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {x:monde_mc.x-100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
```

Par exemple, le premier bouton, qui permet de modifier la position de la carte en x, attache sur l'objet bouton plusX_btn un écouteur. Cet écouteur associe à un événement souris de type clic une fonction nommée plus1.

Dans cette fonction, une autre interpolation non identifiée et non typée apparaît et permet de modifier la propriété x de l'objet monde_mc en incrémentant sa position courante avec monde_mc.x, d'une valeur négative de 100 pixels. À chaque clic sur le bouton, la carte est donc déplacée de 100 pixels en animation vers la gauche. Cette animation contient, en outre, une accélération définie par le paramètre ease.

Définition d'une interpolation TweenMax

La structure d'une interpolation TweenMax est la suivante :

- TweenMax.direction(symbole, durée, {propriété, propriété, propriété..., delay, ease})
- TweenMax. Rappelle le nom de la classe importée pour générer une interpolation.
- Direction. Indique si l'interpolation doit utiliser les paramètres qui suivent pour démarrer l'animation et terminer sur les propriétés courantes de l'objet dans la scène (dans ce cas, nous inscrivons from), ou partir de la position courante de l'objet dans la scène et terminer l'interpolation sur les valeurs passées en paramètre (dans ce cas nous écrivons le terme to).
- Symbole. Indique le nom de l'objet à animer. Par exemple monde_mc.
- Durée. Indique la durée de l'animation en secondes. Pour une durée plus courte, utilisez des valeurs décimales. La valeur 0.5 mentionne une demie seconde.
- Propriété. Entre les deux accolades, vous pouvez ajouter autant de propriétés que vous le souhaitez. Leur ordre n'a pas d'incidence. Chaque propriété doit être séparée par une virgule. Notez que certaines propriétés adoptent une syntaxe particulière comme le tracé Bézier qui place une série de valeur dans une structure tabulaire composée de crochets et où chaque paire de valeurs placée entre accolades est séparée à nouveau par une virgule.
- Delay. Permet d'indiquer un temps avant d'exécuter l'animation. Par exemple, une valeur de 2 retarde le lancement de l'interpolation de deux secondes. L'ordre d'apparition du délai par rapport aux autres propriétés n'a pas d'incidence.
- Ease. Définit le type de transition comme vu dans la section consacrée à la classe Tween. L'ordre d'apparition du paramètre ease par rapport aux autres propriétés n'a pas d'incidence.

Ce principe est également appliqué, sur chaque bouton disponible dans la scène, pour les propriétés y et rotation. Nous obtenons :

```
// Abscisse x
plusX_btn.addEventListener(MouseEvent.CLICK,plus1);
```

```

function plus1 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {x:monde_mc.x-100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
moinsX_btn.addEventListener(MouseEvent.CLICK,moins1);
function moins1 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {x:monde_mc.x+100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}

// Ordonnée y
plusY_btn.addEventListener(MouseEvent.CLICK,plus2);
function plus2 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {y:monde_mc.y-100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
moinsY_btn.addEventListener(MouseEvent.CLICK,moins2);
function moins2 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {y:monde_mc.y+100, delay:0, ease:Strong.easeOut});
    restaurer_btn.visible=true;
}

// Rotation
plusRotation_btn.addEventListener(MouseEvent.CLICK,plus3);
function plus3 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {rotation:monde_mc.rotation-15, delay:0,
    ➤ ease:Strong.easeOut});
    restaurer_btn.visible=true;
}
moinsRotation_btn.addEventListener(MouseEvent.CLICK,moins3);
function moins3 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {rotation:monde_mc.rotation+15, delay:0,
    ➤ ease:Strong.easeOut});
    restaurer_btn.visible=true;
}

```

Les propriétés alpha, scaleX et scaleY sont gérées un peu différemment :

```

// Alpha
plusAlpha_btn.addEventListener(MouseEvent.CLICK,plus4);
function plus4 (evt:MouseEvent) {
    if (monde_mc.alpha<1) {
        TweenMax.to(monde_mc, 1, {alpha:monde_mc.alpha+0.1, delay:0,
        ➤ ease:Strong.easeOut});
        restaurer_btn.visible=true;
    }
}
moinsAlpha_btn.addEventListener(MouseEvent.CLICK,moins4);
function moins4 (evt:MouseEvent) {
    if (monde_mc.alpha>0) {
        TweenMax.to(monde_mc, 1, {alpha:monde_mc.alpha-0.1, delay:0,
        ➤ ease:Strong.easeOut});
        restaurer_btn.visible=true;
    }
}

// Echelle
plusEchelle_btn.addEventListener(MouseEvent.CLICK,plus5);
function plus5 (evt:MouseEvent) {
    TweenMax.to(monde_mc, 2, {scaleX:monde_mc.scaleX*0.5,
    ➤ scaleY:monde_mc.scaleY*0.5, delay:0, ease:Strong.easeOut});
}

```

```

        restaurer_btn.visible=true;
    }
    moinsEchelle_btn.addEventListener(MouseEvent.CLICK,moins5);
    function moins5 (evt:MouseEvent) {
        TweenMax.to(monde_mc, 2, {scaleX:monde_mc.scaleX-0.5,
        ➤ scaleY:monde_mc.scaleY-0.5, delay:0, ease:Strong.easeOut});
        restaurer_btn.visible=true;
    }

```

La gestion de l'alpha est astreinte à ses limites qui sont 0 et 1 afin d'éviter que l'utilisateur n'incrémente trop la valeur dans un sens ou dans l'autre, au-delà de ces limites. Cela évite, lorsqu'il active le bouton opposé, qu'il ne perçoive l'inversion de l'effet avec un retard dû au rythme imposé du pas d'incréméntation.

La gestion de l'échelle adopte quant à elle une interpolation où les propriétés `scaleX` et `scaleY` sont associées et animées simultanément – ce que ne nous permettait pas la classe Tween native, du moins en une seule ligne de code.

Pour terminer, enfin, un bouton de restauration (`restaurer_btn`) initialise tous les paramètres en reprenant les valeurs courantes de l'objet sur la scène, à travers une nouvelle interpolation :

```

// Restaurer
restaurer_btn.addEventListener(MouseEvent.CLICK,restaurerCarte);
function restaurerCarte (evt:MouseEvent) {
    tweenMonde= TweenMax.to(monde_mc, 5, {scaleX:1, scaleY:1, x:400, y:270,
    ➤ rotation:1, alpha:1, delay:0, ease:Strong.easeInOut});
    tweenMonde.addEventListener(TweenEvent.COMPLETE, tweenFin1);
}
function tweenFin1 (evt:TweenEvent) {
    restaurer_btn.visible=false;
}

```

Cette interpolation, à la différence des autres, utilise le nom d'occurrence que nous avons typée dans l'initialisation au début du code. Cela permet de réutiliser cet identifiant pour y attacher, à la suite, un écouteur et lancer l'exécution d'une nouvelle fonction une fois l'animation terminée. En l'occurrence, la fonction invoquée à l'issue de l'interpolation `tween-Monde` restaure la visibilité du bouton.

Enchaîner des actions à une interpolation TweenMax

Pour ajouter des actions suite à une interpolation TweenMax, il est possible d'invoquer une autre fonction *via* un gestionnaire d'événements, comme dans cet exemple :

```
tweenMonde.addEventListener(TweenEvent.COMPLETE, tweenFin1);
```

Vous pouvez aussi invoquer directement une fonction, en paramètre de l'interpolation, comme suit :

```

tweenMonde= TweenMax.to(monde_mc, 5, {scaleX:1, scaleY:1,
x:400, y:270, rotation:1, alpha:1, delay:0, ease:Strong.easeInOut,
➤ onStartListener:fonctionDemarrage, onCompleteListener:fonctionFin });

```

Le paramètre `onStartListener` appelle une nouvelle fonction au démarrage de l'interpolation. Le paramètre `onCompleteListener` appelle une fonction une fois l'interpolation achevée.

À retenir

- La classe TweenMax est plus puissante que la classe Tween native car elle permet d'intégrer plusieurs animations de propriété dans une même instruction et rend ainsi le document plus stable à l'exécution.
- Il est possible, avec TweenMax, d'enchaîner plusieurs animations en y associant la classe TweenEvent, suivie de l'événement MOTION_FINISH ou en passant le nom de la fonction directement en paramètre de l'interpolation à l'aide de la propriété onStartListener ou onCompleteListener.
- Les interpolations de la classe TweenMax permettent d'animer les propriétés d'objets, mais aussi d'utiliser des trajectoires de type Bézier et de déterminer un délai pour retarder l'exécution de l'animation.
- Il est possible de définir une transition depuis la position courante vers une nouvelle position ou inversement, en utilisant to ou from en amorce de la création d'une interpolation TweenMax.
- Un assistant est disponible avec la classe TweenMax. Il permet de s'affranchir de la saisie du code pour réaliser des interpolations plus complexes et explorer d'autres propriétés.

Défilant TweenMax avec un pointeur et un masque

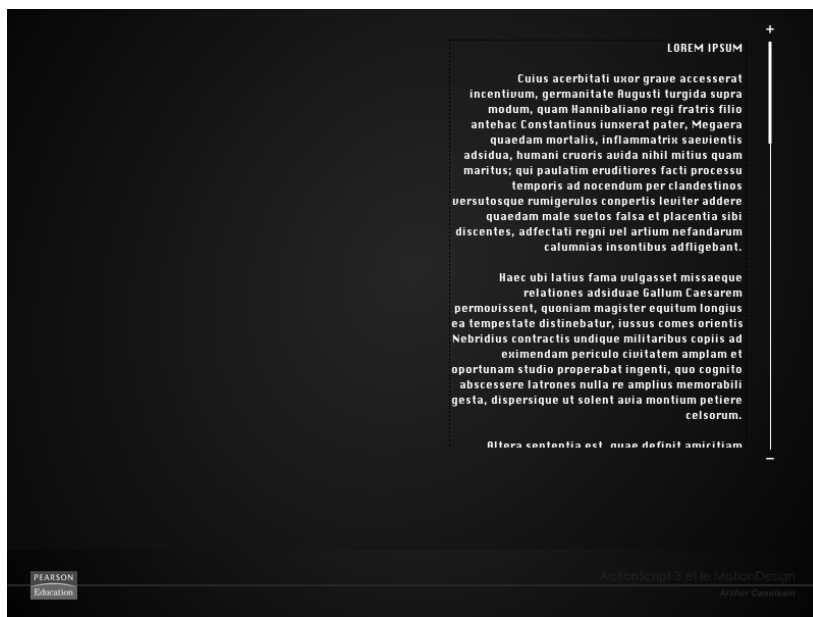
Les classes TweenMax contribuent largement à améliorer l'ergonomie d'un projet en lui offrant des solutions graphiques efficaces pour accompagner le déploiement des contenus dans l'interface d'un site. Elles facilitent l'organisation spatiale des données en offrant la possibilité de scénariser leur mise en forme et servir des fonctionnalités applicatives. Une des applications les plus courantes en ce sens est le contrôle de l'affichage d'un contenu à hauteur variable par une barre d'ascenseur personnalisée.

Dans cette section, nous abordons l'utilisation des classes TweenMax dans le contexte du déploiement d'une barre d'ascenseur personnalisée et adaptable à tout type de contenu. Un ascenseur permet ici de faire défiler un texte avec un gestionnaire de type Event.ENTER_FRAME, mais des boutons plus et moins disposés de part et d'autre de la barre d'ascenseur contrôlent le repositionnement de celui-ci à l'aide des classes TweenMax. Dans cet exemple, nous abordons aussi le déplacement de la barre qui sert à temporiser le défilement du texte, avec la méthode startDrag (voir Figure 2.10).

Le document que nous utilisons présente la structure suivante : dans la scène principale du document, nous distinguons, au-dessus du calque fond_mc, un symbole ascenseur_mc et un texte masqué (voir Figure 2.11). Le symbole ascenseur_mc est composé de plusieurs autres symboles de type MovieClip. Deux symboles : plus_btn et moins_btn, permettent de déplacer le texte verticalement par à-coups successifs dans un sens ou dans l'autre. Un symbole barre_btn permet de le positionner directement à l'endroit souhaité en déplaçant simplement cet objet verticalement. Chacun des symboles utilisés pour le contrôle du défilement du texte contient, à son tour, en plus de la forme graphique qui le caractérise, un symbole de type MovieClip rectangulaire et transparent, plus large que la forme graphique visible rectangle_mc. Cette astuce permet d'étendre les fonctionnalités du symbole au-delà de la surface visible, ce qui est particulièrement intéressant dans ce contexte où les éléments graphiques demeurent très étroits et où l'interaction avec l'objet est la base de l'accès au contenu que nous affichons. Dans chacun des symboles, nous pouvons enfin relever la position de leurs centres respectifs, déterminante pour le calcul du positionnement des objets avec la barre de défilement (voir Figure 2.12).

Figure 2.10

Aperçu
du résultat.

**Figure 2.11**

Aperçu
de la fenêtre
scénario.

**Figure 2.12**

Le positionne-
ment du centre
est situé au
sommet de
chaque symbole.





Exemples > ch2_interpolationEtInteractiviteAvecClassesTweenEtTweenMax_3 fla

Dans le calque actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

var positionInitialTexteY:Number=texte_txt.y;
var hauteurScroll:Number=ascenseur_mc.filetMC_mc.height;
var hauteurBarre:Number=ascenseur_mc.barre_btn.height;
var pas:Number=40;

//----- actions

//// calcul de la position du texte
stage.addEventListener(placerTexte (evt:Event) {
    texte_txt.y=((ascenseur_mc.barre_btn.y*-1)*(texte_txt.height/
    ➤ (hauteurScroll-hauteurBarre))+positionInitialTexteY);
})

//// actions boutons
// +
ascenseur_mc.plus_btn.addEventListener(MouseEvent.CLICK,scrollPlus);
function scrollPlus (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME,placerTexte);
    if (ascenseur_mc.barre_btn.y>=pas) {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:ascenseur_mc.barre_btn.y-pas,
        ➤ ease:Strong.easeOut});
    } else {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:0, ease:Strong.easeOut});
    }
}

// -
ascenseur_mc.moins_btn.addEventListener(MouseEvent.CLICK,scrollMoins);
function scrollMoins (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME,placerTexte);
    if (ascenseur_mc.barre_btn.y<hauteurScroll-hauteurBarre-pas) {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:ascenseur_mc.barre_btn.y+pas,
        ➤ ease:Strong.easeOut});
    } else {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:hauteurScroll-hauteurBarre,
        ➤ ease:Strong.easeOut});
    }
}

// scroll
var zoneDeDeplacement:Rectangle = new Rectangle(0,0,0,hauteurScroll-hauteurBarre);

ascenseur_mc.barre_btn.addEventListener(MouseEvent.CLICK, activer-Deplacement);
function activerDeplacement(evt:MouseEvent){
    addEventListener(Event.ENTER_FRAME,placerTexte);
    evt.currentTarget.startDrag(false,zoneDeDeplacement);
}
```

```

addEventListener(MouseEvent.MOUSE_UP, stopperDeplacement);
function stopperDeplacement(evt:MouseEvent){
    evt.currentTarget.stopDrag();
}

```

Tout d'abord, nous importons les classes requises pour la gestion des interpolations Tween-Max :

```

//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

```

Puis, nous déclarons certaines variables afin de stocker des valeurs et surtout, simplifier le programme en les centralisant ici :

```

var positionInitialTexteY:Number=texte_txt.y;
var hauteurScroll:Number=ascenseur_mc.filetMC_mc.height;
var hauteurBarre:Number=ascenseur_mc.barre_btn.height;
var pas:Number=40;

```

La première variable `positionInitialTexteY` récupère la position courante en y de l'objet texte amené à défiler. Nous obtenons une valeur de 30. En réalité, il s'agit ici de stocker en mémoire le décalage obtenu entre le haut du document et la position limite supérieure que le texte doit atteindre. Nous utilisons cette valeur plus loin pour recalculer le positionnement du texte dont la position se détermine selon la barre de défilement qui démarre, elle, à 0 pixel.

La deuxième variable `hauteurScroll` enregistre la hauteur effective du filet contenu dans l'ascenseur. Nous ne prenons que la hauteur du filet, car celle de l'ascenseur est faussée. L'ascenseur est effectivement plus haut qu'il n'y paraît à cause de la présence occulte des symboles rectangles masqués, qui débordent de part et d'autre à l'intérieur des boutons plus et moins.

La troisième variable `hauteurBarre` enregistre la hauteur de l'objet `barre_mc` qui figure dans l'ascenseur.

Enfin, la dernière variable, `pas`, détermine un pas d'incrémement utilisé dans les animations TweenMax, gérées par les boutons plus et moins.

Pour la plupart des variables, nous avons choisi de définir des valeurs par rapport à des expressions plutôt que par des chiffres, car cela rend le développement naturellement plus adaptable à tout type de configuration. Vous pouvez ainsi récupérer le document et redessiner les formes des boutons ou de la barre, et modifier la hauteur du texte ou le remplacer par un autre objet, le dispositif continuera de fonctionner.

Ensuite, apparaissent les actions :

```

//----- actions

//// calcul de la position du texte
function placerTexte (evt:Event) {
    texte_txt.y=((ascenseur_mc.barre_btn.y*-1)*(texte_txt.height/
    ➤ (hauteurScroll-hauteurBarre))+positionInitialTexteY);
}

```

Dans les actions, nous commençons par créer la fonction qui sera appelée par chacun des boutons plus ou moins, lorsque l'utilisateur cliquera dessus, et uniquement à ce moment-là.

Dans cette fonction, nous déterminons la position de l'objet `texte_txt` en `y`. Par soucis d'ergonomie, nous devons faire descendre le texte lorsque le curseur `barre_mc` monte et inversement. Pour cela, nous devons au préalable considérer que seule la zone utile de l'ascenseur doit être prise en compte dans le calcul, c'est-à-dire la zone située entre l'extrémité du symbole `barre_mc` en mouvement et l'extrémité de la zone de déplacement matérialisée par le filet (voir Figures 2.13 et 2.14).

Nous indiquons donc que la position `y` du texte (`texte_txt.y`) est l'inverse (multipliée par -1) de la position courante de la barre de défilement (`ascenseur_mc.barre_mc.y`). Mais, comme le texte est bien plus grand que ne l'est la zone de défilement, nous devons ajouter un coefficient pour démultiplier la position courante de la barre de déplacement par une valeur qui rende la hauteur de la zone de défilement de l'ascenseur proportionnelle à la hauteur du texte lui-même. Nous multiplions donc la position inversée de la barre par : la hauteur du texte, divisée par la différence entre la hauteur de la zone de défilement et celle de la barre elle-même (`texte_txt.height / (hauteurScroll - hauteurBarre)`).

Pour éviter enfin que le défilement ne remplace le texte à 0 lorsque la barre est située tout en bas, nous ajoutons en plus la valeur `positionInitialTexteY`, stockée initialement, qui détermine le décalage entre le sommet et la position courante du bloc de texte.

Formulée de manière générique, nous obtenons l'équation suivante :

$$\text{positionObjetY} = \text{positionBarreAscenseurY} * \text{CoefficientInversé} * \text{coefficientProportionHauteurTexte} / (\text{HauteurZoneVisible} + \text{HauteurBarreDefilement})$$

Figure 2.13

Positionnement du texte lorsque la barre d'ascenseur est en haut.

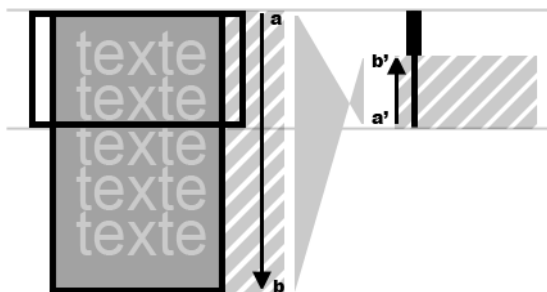
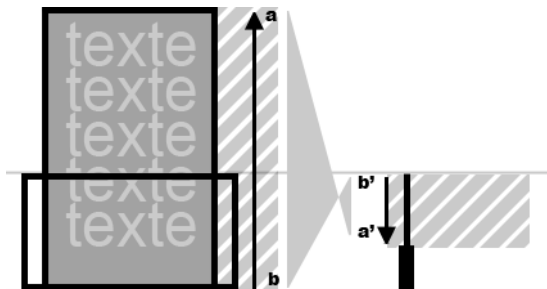


Figure 2.14

Positionnement du texte lorsque la barre d'ascenseur est en bas.





Gérer un ascenseur avec une accélération. Dans le cadre du développement de la barre de défilement, comme vu Chapitre 1, vous pouvez aussi définir un déplacement du contenu avec un coefficient d'accélération au lieu de le déplacer en temps réel par rapport à la position du curseur. Pour cela, vous devez incrémenter, à la position courante de l'objet à rendre mobile (ici, `texte_txt`), la valeur obtenue par notre calcul (expression actuellement utilisée) moins la position courante du contenu (`texte_txt.y`) ; le tout, divisé par une valeur comprise entre 1 et 10 grand maximum. Reste alors à jauger le coefficient de repositionnement de l'objet `texte_txt` de sorte qu'il ne parte pas dans le décor lorsque le curseur `barre_mc` atteint les extrémités de la zone de défilement.

Ensuite, apparaît l'action du bouton `plus_btn` :

```
//// actions boutons
// +
ascenseur_mc.plus_btn.addEventListener(MouseEvent.CLICK,scrollPlus);
function scrollPlus (evt:MouseEvent) {
    if (ascenseur_mc.barre_btn.y>=pas) {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:ascenseur_mc.barre_btn.y-pas,
        ➡ ease:Strong.easeOut});
    } else {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:0, ease:Strong.easeOut});
    }
}
```

Lorsque l'utilisateur presse le bouton de la souris sur l'objet `plus_btn`, la fonction `scrollPlus` s'exécute. Dans cette fonction, nous créons deux interpolations `TweenMax`. La première qui incrémente la position de la barre d'un pas de 40 pixels est lancée si et seulement si cette barre d'ascenseur atteint une position supérieure ou égale à 40 pixels. Au-dessous, si nous continuions d'incrémenter sa position de 40 pixels, la barre dépasserait la limite de l'ascenseur définie par le sommet du filet.

Lorsque la barre se trouve à une distance inférieure à un pas d'incrémentation, en revanche, nous créons une animation `TweenMax` qui replace directement la barre à la position 0 qui correspond au sommet du filet avec `else`.

Pour le bouton `moins_btn`, nous déclinons les valeurs en les inversant et nous obtenons le code suivant :

```
// -
ascenseur_mc.moins_btn.addEventListener(MouseEvent.CLICK,scrollMoins);
function scrollMoins (evt:MouseEvent) {
    if (ascenseur_mc.barre_btn.y<hauteurScroll-hauteurBarre-pas) {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:ascenseur_mc.barre_btn.y+pas,
        ➡ ease:Strong.easeOut});
    } else {
        TweenMax.to(ascenseur_mc.barre_btn, 1, {y:hauteurScroll-hauteurBarre,
        ➡ ease:Strong.easeOut});
    }
}
```

Le bouton `moins_btn` adopte le même calcul que pour le bouton `plus_btn`, à la différence que le point limite de repère est défini par la hauteur de l'ascenseur moins la hauteur de la barre, moins le pas d'incrémentation. Cela correspond, compte tenu de la hauteur effective de la barre de défilement (voir Figure 2.12), à la limite inférieure du filet moins le pas

d'incréméntation. Si cette limite est dépassée, alors (avec `else`), l'animation `TweenMax` déplace la barre jusqu'à l'extrémité inférieure du filet.

La gestion du déplacement de la barre de défilement, enfin, se définit par le code suivant :

```
// scroll
var zoneDeDeplacement:Rectangle = new Rectangle(0,0,0,hauteurScroll-hauteurBarre);

ascenseur_mc.barre_btn.addEventListener(MouseEvent.CLICK, activerDeplacement);
function activerDeplacement(evt:MouseEvent){
    addEventListener(Event.ENTER_FRAME, placerTexte);
    evt.currentTarget.startDrag(false, zoneDeDeplacement);
}

addEventListener(MouseEvent.CLICK, stopperDeplacement);
function stopperDeplacement(evt:MouseEvent){
    evt.currentTarget.stopDrag();
}
```

D'abord, nous définissons les limites de la zone de défilement pour le symbole `barre_btn`. Cette limite est définie à travers la création d'une variable qui stocke les coordonnées géométriques d'un nouvel objet `Rectangle`. Le typage `:Rectangle` gère la mémoire requise pour stocker les coordonnées. S'en suit la création physique de l'objet avec ses coordonnées :

```
var zoneDeDeplacement:Rectangle = new Rectangle(0,0,0,hauteurScroll-hauteurBarre);
```

Une zone de déplacement se définit à travers une forme géométrique. L'équation qui permet de définir les coordonnées d'un rectangle en AS3 est : `Rectangle (x1, y1, largeur, hauteur)`. En choisissant une largeur ou une hauteur nulle, nous limitons le défilement à un axe vertical ou horizontal. Dans notre exemple, la hauteur correspond à celle du filet moins celle de la barre, car le centre de la barre est situé au sommet de cette dernière (voir Figure 2.12). Il faut donc soustraire le différentiel qui reste entre la position de son centre et sa hauteur.

Ensuite, lorsque l'utilisateur presse le bouton de la souris sur la barre, un écouteur exécute la fonction `zoneDeDeplacement` qui active le déplacement du texte. Ce déplacement est amorcé à la fois avec avec la fonction `startDrag` qui permet de rendre mobile le curseur `barre_mc` et en appelant la fonction déjà identifiée (`placerTexte`), qui recalcule la position du texte en fonction de la position du curseur :

```
ascenseur_mc.barre_btn.addEventListener(MouseEvent.CLICK, activerDeplacement);
function activerDeplacement(evt:MouseEvent){
    addEventListener(Event.ENTER_FRAME, placerTexte);
    evt.currentTarget.startDrag(false, zoneDeDeplacement);
}
```

Nous appelons, seulement maintenant, la fonction `placerTexte` car elle calcule de manière perpétuelle la position courante du texte et de ce fait, sollicite de manière continue les ressources de l'ordinateur. Pour optimiser ces ressources, nous n'activons la fonction que lorsqu'elle est nécessaire, et non en plaçant l'écouteur à l'exécution de l'animation, comme en l'isolant hors de la fonction par exemple. De même, nous pourrions stopper la fonction lorsque le pointeur relâche le curseur, mais ce faisant, nous perdriions l'effet d'amortissement dans le mouvement du texte si l'utilisateur activait plusieurs fois de suite l'un des deux boutons plus ou moins, avant que l'animation ne se termine, même avec une fonction

retardée suite à un événement de type `TweenEvent.COMPLETE`. Nous préférons donc garder cette fonction active.



Arrêter un écouteur. Pour arrêter un écouteur actif, nous utilisons la méthode `removeEventListener`, suivie du même gestionnaire d'événements que celui utilisé pour activer la fonction à neutraliser. Ainsi, pour stopper l'écouteur :

```
addEventListener(Event.ENTER_FRAME,nomDeLaFonction)
```

nous inscrivons :

```
removeEventListener(Event.ENTER_FRAME,nomDeLaFonction).
```

Également dans notre fonction, nous activons le déplacement avec la méthode `startDrag()`. En paramètre, nous déterminons d'abord si l'objet déplacé reste collé au pointeur en son centre (`true`) ou non (`false`). Puis, nous rappelons les coordonnées de l'objet rectangle créé en amont, afin de restreindre le déplacement aux limites géométriques du rectangle.

Le programme s'achève avec un gestionnaire associé à l'événement `MOUSE_UP` pour arrêter le déplacement du curseur `barre_btn`, et donc, le défilement du texte, en exécutant la fonction `stopperDeplacement` :

```
addEventListener(MouseEvent.CLICK, stopperDeplacement);  
function stopperDeplacement(evt:MouseEvent) {  
    evt.currentTarget.stopDrag();  
}
```

La fonction indique d'arrêter le déplacement avec la méthode `stopDrag`, lorsque la souris est relâchée, en ciblant l'objet courant actif avec `evt.currentTarget` (pour en savoir plus sur la différence entre `target` et `currentTarget`, consultez le Chapitre 5 où un exemple illustre le contexte d'utilisation de la propriété `target`).

À retenir

- Utiliser des expressions plutôt que des valeurs permet de rendre le développement adaptable à tout type de mise en forme.
- La classe `TweenMax` peut être combinée à des animations gérées avec la classe `Event.ENTER_FRAME` si les objets contrôlés par l'interpolation ne sont pas gérés directement par la classe `Event`.
- Il est possible d'étendre la zone réactive d'un objet en utilisant des symboles cachés auxquels nous appliquons un `alpha nul` ou en créant un bouton pour lequel seule la zone cliquée possède une forme pleine.
- Pour gérer un déplacement d'objet, nous utilisons les méthodes `startDrag` et `stopDrag`, que l'on associe aux limites d'une forme géométrique comme un rectangle pour contraindre le déplacement.
- Pour définir un déplacement rectiligne, il suffit de rendre nulle soit la largeur, soit la hauteur du rectangle utilisé comme zone limite de déplacement.
- Pour calculer les rapports de valeurs entre les propriétés de différents objets, il faut bien prendre en compte la structure de l'objet référent (la position de son centre, sa hauteur, les éléments visibles ou invisibles qui le composent, son décalage par rapport à la scène courante, etc).

Synthèse

Dans ce chapitre, vous avez appris à réaliser des interpolations avec des classes extrêmement puissantes. Elles offrent la possibilité, en une ligne de code, de gérer de nombreuses propriétés, une trajectoire et un délai d'exécution notamment. Vous êtes à présent en mesure d'explorer également d'autres classes disponibles dans l'assistant TweenMax et vous savez désormais les intégrer par défaut à toutes vos applications, sans avoir à les ajouter manuellement dans chacun de vos nouveaux projets.

3

Les transitions d'effets et de filtres

Introduction

Les transitions permettent d'ajouter aux animations des effets sur les contenus (fondus enchaînés, Flash d'appareil photo, halo lumineux, flou directionnel, volets ou stores ouvrants, etc.). Pour permettre la gestion de ces effets dans le temps, nous utilisons des classes spécifiques, qui sont `transitionManager` pour les effets et `filters` pour les filtres, et les associons éventuellement à un chronomètre de type `Timer`. En typant ces transitions, il est en outre possible de les faire suivre d'autres effets animés et d'organiser une mise en scène graphique des contenus sur la durée avec des variations d'effets.

Dans ce chapitre, nous présentons deux galeries photos qui exploitent respectivement les effets et les filtres en les combinant à des chronomètres.

Galerie photo avec transition d'effets

Dans cette section, nous abordons les différents types d'effets programmables en ActionScript 3. Pour cela, nous appliquons dans un premier exemple un effet de fondu sur une galerie d'images, puis nous détaillons les propriétés des autres effets disponibles à partir de la classe `TransitionManager`.

Effet de fondu avec un Timer

Dans cet exemple, une galerie fait apparaître, en fondu, une série de photographies intégrées physiquement dans la scène du document Flash. Vous pouvez modifier les images et leur nombre ainsi que la durée et le style des transitions, pour personnaliser ce script selon votre convenance et l'intégrer à nos projets (voir Figure 3.1).

Figure 3.1

Aperçu
du document.





Exemples > ch3_transitionsEffetsEtFiltres_1 fla

Le document que nous utilisons présente la structure suivante : dans la scène, au-dessus du calque `fond_mc`, apparaît un symbole nommé `galerie_mc` (voir Figure 3.2). Ce symbole contient une suite d'images toutes différentes. Une action `stop()` empêche la tête de lecture de jouer l'animation dès l'exécution du document Flash (voir Figure 3.3).

Figure 3.2

Aperçu du scénario de la scène principale.

**Figure 3.3**

Aperçu du scénario du symbole `galerie_mc`.



Sur la première image du calque nommé `actions`, figure le code suivant :

```
//----- initialisation
import fl.transitions.*;
import fl.transitions.easing.*;

var i:Number=1;
var nombreDePhotos:Number=6;
var dureeBoucle:Number=8000;
var boucle:Timer=new Timer(dureeBoucle,nombreDePhotos);
var transitionPhoto:TransitionManager=new TransitionManager(galerie_mc);

galerie_mc.visible=false;

//----- actions

boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
boucle.start();

function lancerBoucle(evt:TimerEvent) {
    lancerGalerie();
}

function lancerGalerie () {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        transitionPhoto.startTransition({type:Fade, direction:Transition.IN,
            ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
        transitionPhoto.addEventListener("allTransitionsInDone", suite);
        function suite(evt:Event) {
            transitionPhoto.startTransition({type:Fade, direction:Transition.OUT,
                ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
        }
        i++;
    } else {
```

```

        galerie_mc.visible=false;
    }
}
lancerGalerie();

transitionPhoto.addEventListener("allTransitionsOutDone", suite2);
function suite2(evt:Event) {
    MovieClip(evt.target.content).visible=false;
}

```

Tout d'abord, nous importons les classes nécessaires pour l'animation :

```

//----- initialisation
import fl.transitions.*;
import fl.transitions.easing.*;

```

Ensuite, nous définissons les variables :

```

var i:Number=1;
var nombreDePhotos:Number=6;
var dureeBoucle:Number=8000;

```

La variable `i` sert ici à définir la position de la tête de lecture sur l'image du scénario du symbole `galerie_mc` pour chaque nouvelle image à afficher.

La variable `nombreDePhotos` permet de déterminer le nombre de photos à jouer. Cette valeur correspond *a priori* au nombre d'images qui figurent dans le scénario du clip `galerie_mc`, mais elle peut être inférieure.

La variable `dureeBoucle` indique la durée d'une animation pour chaque photo, effets de transition inclus. Cette durée, comme nous le précisons plus loin, est intégrée dans le calcul de la durée de chaque effet. Ainsi, plus la durée de l'animation d'une image est longue, plus les transitions seront longues également. La durée est exprimée en millisecondes. Une valeur de 8 000 correspond donc à un changement d'image toutes les 8 secondes.

Ensuite, une variable `boucle` déclare la création d'un chronomètre. Ce chronomètre enclenche la répétition d'une action autant de fois qu'il y a d'images déclarées (`nombreDePhotos`), donc toutes les 8 secondes (`dureeBoucle`). Nous pourrions définir le chronomètre selon l'expression suivante : `Timer(durée, nombre de répétition)`.

```

var boucle:Timer=new Timer(dureeBoucle,nombreDePhotos);

```

Puis, nous déclarons une nouvelle transition :

```

var transitionPhoto:TransitionManager=new TransitionManager(galerie_mc);

```



Calcul de la durée d'un chronomètre. Lorsque vous spécifiez une durée sur un `Timer`, comme suit : `Timer(8000,5)`, notez que la fonction appelée par l'écouteur, attaché au `Timer`, est activée dans cet exemple toutes les 8 secondes et 5 fois de suite. Les interpolations exécutées au sein de la fonction appelée, par conséquent, ne doivent pas durer ici plus de 8 secondes. Elles doivent même durer légèrement moins que 8 secondes, environ 7 990 milli-secondes. Car le lecteur Flash requiert quelques millièmes de secondes pour lancer une action. En prévoyant une durée trop juste, vous risquez de dépasser la durée prévue et de rendre l'exécution de l'animation instable.

Le symbole `galerie_mc` qui contient les images est passé en paramètre et c'est lui qui sera affecté par les effets de la classe `transitionManager`. Enfin, la galerie est masquée par défaut :

```
galerie_mc.visible=false;
```

C'est seulement lorsque la fonction `lancerGalerie` sera active que la galerie sera réaffichée. Viennent ensuite les actions :

```
//----- actions

boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
boucle.start();

function lancerBoucle(evt:TimerEvent) {
    lancerGalerie();
}
```

D'abord, un écouteur est attaché au chronomètre afin de capter les itérations et de programmer une action à chacune d'entre elles. Pour cela, nous utilisons la classe `TimerEvent.TIMER` qu'il n'est pas nécessaire d'importer au préalable car, comme la classe `Event` ou `MouseEvent`, elle est incluse nativement dans l'API du logiciel. Cet écouteur exécute alors une fonction nommée `lancerBoucle`.

À son tour, la fonction `lancerBoucle` appelle une autre fonction nommée `lancerGalerie`.

Pourquoi ne pas directement appeler les instructions de la fonction `lancerGalerie`, lors de la première fonction `lancerBoucle` ? Cette première fonction est un écouteur de type chronomètre et de ce fait elle ne sera exécutée qu'après un certain laps de temps. Si nous appelions directement les instructions depuis la fonction `lancerBoucle`, la galerie ne serait lancée qu'au bout de 8 secondes. Il faudrait alors attendre que les 8 premières secondes soient écoulées avant que la première transition ne soit exécutée. Pour permettre de jouer la galerie dès l'instant 0, nous devons pouvoir exécuter cette fonction initialement, indépendamment du chronomètre. Pour cela, nous devons isoler les instructions dans une nouvelle fonction, autonome et distincte, qui n'attend pas de paramètre d'objet, comme `lancerGalerie()`. Nous pouvons alors appeler cette fonction, à la fois depuis notre chronomètre avec la fonction `lancerBoucle(evt:TimerEvent)` et directement à l'initialisation de l'image du scénario, avec l'instruction `lancerGalerie()`. Le premier appel gère l'exécution de la galerie à partir de 8 secondes jusqu'à la fin de l'animation. Le second appel l'exécute dès la publication du document. Comme la fonction est terminée avant 8 secondes, les deux appels s'enchaînent parfaitement et l'incrémentation activée dès la première exécution continue de croître avec le chronomètre dans les suivantes.

Puis, nous créons la fonction qui exécute la galerie animée :

```
function lancerGalerie () {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        transitionPhoto.startTransition({type:Fade, direction:Transition.IN,
        ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
        transitionPhoto.addEventListener("allTransitionsInDone", suite);
        function suite(evt:Event) {
            transitionPhoto.startTransition({type:Fade, direction:Transition.OUT,
            ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
        }
    }
}
```

```

        i++;
    } else {
        galerie_mc.visible=false;
    }
}

```

Dans cette fonction autonome, dans un premier temps, nous rendons visible la galerie :

```
galerie_mc.visible=true;
```

Puis, nous ajoutons une structure conditionnelle avec incrémentation de la valeur *i* :

```

if (i<=nombreDePhotos) {
}
i++
} else {
    galerie_mc.visible=false;
}

```

Cette structure indique d'exécuter une action tant que la valeur de *i* est inférieure au nombre d'images défini à travers *nombreDePhotos*, dont nous avons déterminé la valeur plus haut à 6. Lorsque *i* atteint le nombre d'images, le chronomètre est terminé et la galerie n'apparaît plus, car à cet instant, nous la rendons invisible.

À cette structure, nous intégrons les instructions pour afficher les images :

```

function lancerGalerie () {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
    }
    i++;
} else {
    galerie_mc.visible=false;
}
}

```

Dans un premier temps, nous spécifions d'atteindre l'image du scénario du symbole *galerie_mc* qui correspond à la valeur de *i*. Par défaut, sa valeur est 1. C'est donc la première photo qui est jouée au lancement de l'animation. Cette valeur est incrémentée plus loin, en dehors du bloc d'instructions avec *i++*, afin de s'assurer que l'incrément fonctionne indépendamment de l'animation. Toutes les 8 secondes donc, y compris dès l'exécution de la fonction avant le chronomètre *Timer*, une nouvelle photo est affichée jusqu'à la dernière où la galerie disparaît complètement.

Nous spécifions alors les transitions à joindre à notre fonction :

```

function lancerGalerie () {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        transitionPhoto.startTransition({type:Fade, direction:Transition.IN,
        ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
    }
    i++;
} else {
    galerie_mc.visible=false;
}
}

```

Nous utilisons une occurrence de la classe `transitionManager` typée plus haut sous le nom `transitionPhoto`. Cette transition démarre avec un fondu (`type:Fade`), en entrée (`direction:Transition.IN`), d'une durée de 4 secondes (`dureeBoucle/2000=4`), et avec une accélération en entrée et en sortie (`easing:Strong.easeInOut`).

À la suite, dans le même bloc d'instruction, nous ajoutons une nouvelle transition qui s'exécute une fois la première transition achevée, grâce à un écouteur associé à la sous-classe `allTransitionsInDone` (voir encadré). Cette nouvelle transition génère un effet de fondu en sortie :

```
function lancerGalerie () {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        transitionPhoto.startTransition({type:Fade, direction:Transition.IN,
        ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
        transitionPhoto.addEventListener("allTransitionsInDone", suite);
        function suite(evt:Event) {
            transitionPhoto.startTransition({type:Fade, direction:Transition.OUT,
            ➤ duration:dureeBoucle/2000, easing:Strong.easeInOut});
        }
        i++;
    } else {
        galerie_mc.visible=false;
    }
}
```

La mise en place de la fonction est maintenant terminée. Nous pouvons l'exécuter au démarrage :

```
lancerGalerie();
```

Définition de la classe *transitionManager*

Une transition de la classe `transitionManager`, appartenant à la famille de la classe `transitions`, peut être lancée de deux manières. Soit nous activons directement une interpolation sans la typer :

```
TransitionManager.start(monClip_mc,{type:Fade, direction:Transition.IN,
➤ duration:4, easing:Strong.easeInOut})
```

Soit nous la typons d'abord afin de permettre son identification par la suite et y associer éventuellement plus tard un écouteur, pour enchaîner avec d'autres animations par exemple :

```
var nomDeLaTransition:TransitionManager = new TransitionManager (galerie_mc);
nomDeLaTransition.startTransition({type:Fade, direction:Transition.IN,
➤ duration:4, easing:Strong.easeInOut});
```

Dans les deux cas, les paramètres de la transition sont les suivants :

```
{type, direction, duration, easing}
```

- `type` désigne le type d'effet à appliquer. Nous distinguons les effets suivants : `Blinds`, `Fade`, `Fly`, `Iris`, `Rotate`, `Photo`, `PixelDissolve`, `Squeeze`, `Wipe` et `Zoom`.
- `Blinds` correspond à un effet de stores horizontaux.
- `Fade` correspond à un effet de fondu.
- `Fly` correspond à un effet de translation.

Chacun de ces effets peut être ajusté en fonction des paramètres suivants :

- Iris correspond à un effet d'ouverture en cercle similaire à un diaphragme d'appareil photo.
- Rotate effectue une rotation.
- Photo lance un flash blanc pour simuler un appareil photo. Cet effet peut, par exemple, être associé à la programmation d'un son de déclenchement d'obturateur.
- PixelDissolve joue une mosaïque de formes carrées pour simuler un effet de décomposition de l'image.
- Squeeze écrase l'image.
- Wipe joue un volet sur l'image comme un masque qui se déplace latéralement.
- Zoom réduit l'image jusqu'à sa disparition.
- direction correspond au point de départ de l'effet. La valeur `Transition.IN` active l'effet en ouverture. `Transition.OUT`, l'active en fermeture.
- duration correspond à la durée de l'effet, en secondes.
- easing, enfin, indique le mode d'accélération à attacher à l'effet (voir Chapitre 2 sur la définition du paramètre easing pour la classe Tween).

Pour détecter la fin de l'interpolation avec `transitionManager`, attachez un écouteur sur le nom d'occurrence de la transition et spécifiez l'événement `allTransitionInDone` si la transition précédemment exécutée est `transition.IN`. Spécifiez à l'inverse `allTransition-OutDone` si la transition précédemment exécutée est `transition.OUT`. Par exemple :

```
nomDeLaTransition.addListener("allTransitionsInDone", suite);
function suite(evt:Event):void {
    nomDeLaTransition.startTransition({type:Fade,
                                        direction:Transition.OUT,
                                        duration:4,
                                        easing:Strong.easeInOut});
}
```



Où placer l'écouteur et la fonction ? L'emplacement de la fonction et de son appel indiffère. Vous pouvez appeler une fonction avant ou après l'avoir rédigée, le lecteur exécutera les deux simultanément. Les designers préfèrent généralement placer la fonction après l'écouteur ou après l'appel de la fonction, car cela reprend une progression logique de l'objet placé dans la scène vers le programme. Tandis que les programmeurs puristes préfèrent souvent placer les fonctions avant les écouteurs, car ils pensent d'abord aux instructions avant leur mise en forme dans la scène qui bien souvent n'existe pas.

Enfin, nous terminons en attachant à nouveau un écouteur à la transition qui suit la deuxième interpolation :

```
transitionPhoto.addListener("allTransitionsOutDone", suite2);
function suite2(evt:Event) {
    MovieClip(evt.target.content).visible=false;
}
```


Cette dernière fonction rend la galerie invisible une fois le fondu sortant achevé. Nous ciblons, pour ce faire, la transition courante, avec `evt.target`, en la convertissant d'abord en `MovieClip`, avec `MovieClip(evt.target.content)`, afin de permettre de lui appliquer ensuite une propriété de `MovieClip` avec `visible=false`.

Le langage

Le transtypage.

La méthode qui consiste à convertir le typage d'un objet en un autre type d'objet se nomme le transtypage. Elle est utilisée pour permettre l'affectation de certaines propriétés et méthodes à des objets que leur type n'autorise initialement pas. Nous parlons aussi d'étendre les propriétés d'un objet, mais cette dernière notion est plutôt réservée à la création de sous-classes personnalisées ajoutées à des objets existants.



Détecter la fin d'une boucle d'itération Timer. Pour détecter la fin d'une boucle `Timer`, attachez au `Timer` un écouteur associé à la classe `Timer.Event.TIMER_COMPLETE` :

```
boucle.addEventListener(TimerEvent.TIMER_COMPLETE,boucleFinie);
function boucleFinie() {
    trace("fin du timer") ;
}
```

Arrêter un Timer. Pour arrêter le déroulement d'un `Timer` en cours d'exécution, utilisez un simple `stop` :

```
boucle.stop();
```

Pour expérimenter la galerie de photos avec d'autres types de transitions, vous pouvez remplacer le type défini dans chaque transition par celui de votre choix, parmi ceux définis dans l'encadré (`Blinds`, `Fade`, `Fly`, `Iris`, `Rotate`, `Photo`, `PixelDissolve`, `Squeeze`, `Wipe`, `Zoom`), et cumuler d'autres transitions à celles-ci pour les agrémenter, par exemple, d'effets supplémentaires.

À retenir

- La classe `transitionManager` est native et permet d'ajouter des effets de transition directement sur des objets de la scène.
- La classe `transitionManager` peut être suivie d'autres animations avec la sous-classe `allTransitionOutDone` ou `allTransitionOutDone` ou avec la sous-classe `TimerEvent.TIMER_COMPLETE` du `Timer`.
- Pour contrôler l'exécution d'une fonction à intervalles réguliers, nous utilisons un `Timer`.
- Un `Timer` exécute chaque itération à la fin de la durée et non au début. Et requiert, en conséquence, d'appeler une première fois la fonction attachée à l'écouteur du `Timer` avant de l'exécuter par le `Timer`.

Galerie photo avec transition de filtres

Dans cette section, nous utilisons la classe `filters` qui permet, selon le même principe que pour la classe `transitionManager`, d'appliquer dynamiquement des filtres à des objets (flou, ombre portée, halo ou biseau).

Pour cela, nous utilisons une galerie de photos qui affiche des vues macroscopiques de planètes à travers une lunette. Chaque transition, d'une image à l'autre, applique simultanément des filtres flou, ombre portée, halo et biseau (voir Figure 3.3). Nous obtenons une galerie d'images où chaque transition est globalement floue mais redevient nette une fois qu'une nouvelle image a été affichée.

Figure 3.3

Aperçu de l'exercice.



Exemples > ch3_transitionsEffetsEtFiltres_2.fla

Le document que nous utilisons présente la structure suivante : dans la scène, au-dessus du calque `fond_mc` (voir Figure 3.4), un symbole nommé `galerie_mc` contient une suite d'images toutes différentes et masquées. À l'intérieur du symbole `galerie_mc`, une action `stop()` empêche la tête de lecture de jouer l'animation (voir Figure 3.5).

Figure 3.4

Aperçu du scénario de la scène principale.

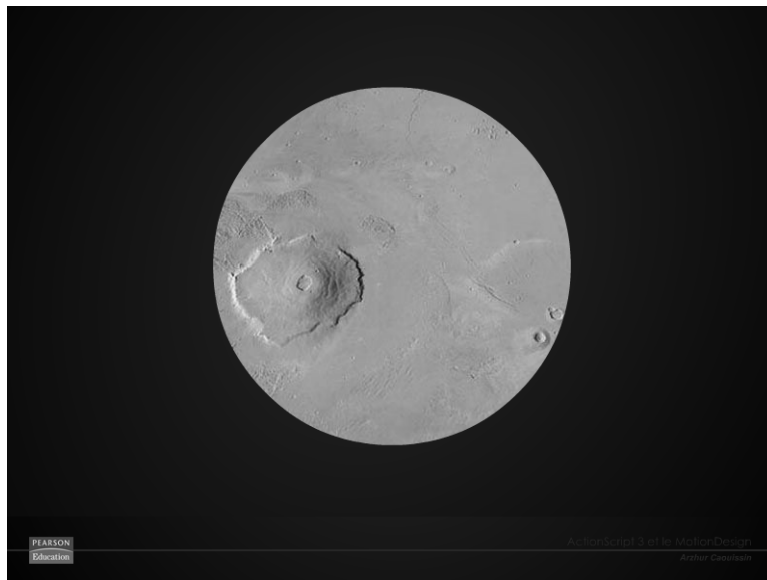
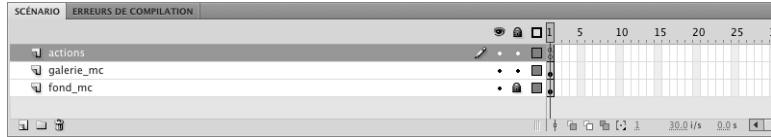


Figure 3.5

Aperçu du scénario
du symbole
galerie_mc.



Sur la première image du calque nommé actions, apparaît le code suivant :

```
//----- initialisation

import fl.transitions.*;
import fl.transitions.easing.*;
import flash.filters.*;

import gs.*;
import gs.easing.*;
import gs.events.*;

var flou:BlurFilter=new BlurFilter(0,0,3);
var ombrePortee:DropShadowFilter=new DropShadowFilter(50, 45, 1, 0x000000,
↳ 20, 10, 1, 3, false, false, false);
var halo:GlowFilter=new GlowFilter(0xffffffff, 1, 20, 20, 3, 255, false, false);
var biseau:BevelFilter=new BevelFilter(50,45,0xFFFFF,1,0x000000,1,120,120,
↳ 1,3,"inner",false);

var i:Number=1;
var nombreDePhotos:Number=5;
var dureeBoucle:Number=10000;
var boucle:Timer=new Timer(dureeBoucle,nombreDePhotos);

galerie_mc.visible=false;

//----- actions

boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
boucle.start();

function lancerBoucle(evt:TimerEvent) {
    lancerGalerie();
}

function lancerGalerie() {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        // INITIALISATION
        galerie_mc.filters=[flou];
        galerie_mc.filters=[ombrePortee];
        galerie_mc.filters=[halo];
        galerie_mc.filters=[biseau];
        //ANIMATION INTRO
        TweenMax.from(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100,
↳ quality:3}, delay:0, ease:Strong.easeInOut});
        TweenMax.from(galerie_mc, 4, {dropShadowFilter:{distance:5, angle:45,
↳ alpha:1, color:0x000000, blurX:10, blurY:10, strength:1,
↳ quality:3, type:"inner", knockout:false, hideObject:false},
↳ delay:0, ease:Strong.easeInOut});
```

```

        TweenMax.from(galerie_mc, 4, {glowFilter:{color:0xffffffff, alpha:1,
        ➤ blurX:0, blurY:0, strength:1, quality:3, type:"inner",
        ➤ knockout:false}, delay:0, ease:Strong.easeInOut});
        TweenMax.from(galerie_mc, 4, {bevelFilter:{distance:5, angle:45,
        ➤ highlightColor:0xffffffff, highlightAlpha:1,
        ➤ shadowColor:0x000000, shadowAlpha:1, blurX:0, blurY:0,
        ➤ strength:1, quality:3, type:"inner", knockout:false},
        ➤ delay:0, ease:Strong.easeInOut});
    //ANIMATION SORTIE
    TweenMax.to(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100, quality:3},
    ➤ delay:5, ease:Strong.easeInOut});
    TweenMax.to(galerie_mc, 4, {dropShadowFilter:{distance:5, angle:45,
    ➤ alpha:1, color:0x000000, blurX:10, blurY:10}, delay:5,
    ➤ ease:Strong.easeInOut});
    TweenMax.to(galerie_mc, 4, {glowFilter:{color:0xffffffff, alpha:1, blurX:0,
    ➤ blurY:0}, delay:5, ease:Strong.easeInOut});
    TweenMax.to(galerie_mc, 4, {bevelFilter:{distance:5, angle:45,
    ➤ color:0x000000, blurX:0, blurY:0}, delay:5, ease:Strong.easeInOut});
    i++;
} else {
    galerie_mc.visible=false;
}
}
lancerGalerie();

```

Nous importons en premier lieu les classes requises :

```

//----- initialisation

import fl.transitions.*;
import fl.transitions.easing.*;
import flash.filters.*;

import gs.*;
import gs.easing.*;
import gs.events.*;

```

Les premières classes importées permettent de réaliser les interpolations (transitions et easing). La classe `filters`, elle, permet l'exploitation des filtres. Les classes et sous-classes `Greensocks` permettent l'utilisation de `TweenMax` pour rendre possible l'animation de ces filtres. Attention, cette classe requiert le dossier `gs` pour importer les classes qui ne sont pas disponibles par défaut dans l'API de Flash (voir Chapitre 2).

Ensuite, nous prédéfinissons quatre filtres que nous exploitons plus loin dans le code :

```

var flou:BlurFilter=new BlurFilter(100,100,3);
var ombrePortee:DropShadowFilter=new DropShadowFilter(50, 45, 1, 0x000000,
➤ 20, 10, 1, 3, false, false, false);
var halo:GlowFilter=new GlowFilter(0xffffffff, 1, 20, 20, 3, 255, false, false);
var biseau:BevelFilter=new BevelFilter(50,45,0xFFFFF,1,0x000000,
➤ 1,50,50,100,3,"inner",false);

```

Tous les paramètres renseignés dans chacun des filtres ne sont pas obligatoires, mais leur ordre d'affichage est important si leur désignation n'est pas renseignée. Par exemple, vous pouvez écrire :

```

var flou:BlurFilter=new BlurFilter(0,0,3);

```

Mais vous pouvez aussi écrire :

```
blurFilter:{blurX:100, blurY:100, quality:3}
```

ou bien :

```
var flou:BlurFilter = new BlurFilter();  
flou.quality = BitmapFilterQuality.MEDIUM;  
flou.blurX = 0;  
flou.blurY = 0;
```

Dans le premier cas, la désignation n'est pas renseignée. L'ordre de listage des valeurs est interchangeable. Dans le second cas (deuxième et troisième exemple), la désignation est renseignée. Vous pouvez modifier l'ordre de listage des propriétés et indiquer en premier la qualité par exemple sans affecter le bon rendu de l'effet.

Filtre flou

Le premier filtre, nommé `flou`, type et crée un nouvel objet filtre flou `BlurFilter(0,0,3)` dont les valeurs indiquent un flou de 0 pixel de diamètre et de qualité optimale sur une échelle de 1 à 3. La valeur du flou est ici portée à zéro car en réalité nous initialisons le filtre flou avant de l'animer.

Il est appliqué, dans la fonction, grâce à la commande suivante :

```
var flou:BlurFilter=new BlurFilter(0,0,3);  
galerie_mc.filters=[flou];
```



Définition des paramètres du filtre flou `BlurFilter`. Le filtre flou se définit selon l'expression suivante :

```
blurFilter:{blurX:100, blurY:100, quality:3}
```

La définition des paramètres du filtre flou sont :

- `blurFilter`, le nom du filtre à appliquer reconnu comme tel par la classe `filters`.
- `blurX`, indique l'étalement du flou en pixels sur l'axe des X. Une valeur de 50 étale le flou sur une distance de 50 pixels en largeur.
- `blurY`, indique l'étalement du flou en pixels sur l'axe des Y. Une valeur de 50 étale le flou sur une distance de 50 pixels en hauteur.
- `quality`, indique la qualité du flou. Cette valeur s'exprime sur une échelle de 1 à 3. 1 correspond à un flou grossier mais moins gourmand en ressources graphiques. 2, à un flou moyen et 3, à un flou fin ou gaussien, plus gourmand en ressources graphiques.

Filtre ombre portée

Le deuxième filtre, nommé `ombrePortee`, type et crée un nouvel objet filtre d'ombre portée `DropShadow(50, 45, 1, 0x000000, 20, 10, 1, 3, false, false, false)` dont les valeurs indiquent respectivement une ombre de 50 pixels de distance par rapport à l'objet, de 45° d'angle, d'un alpha à 1 (soit 100 %), de couleur noire, d'un flou en X et Y respectivement de 20 et 10 pixels, d'une force de 1 sur une échelle de 1 à 255, de qualité 3, avec une option ombre intérieure inactive, un masque inactif, et un cache sur l'objet inactif).

Le filtre est appliqué, plus tard dans la fonction, grâce à la commande suivante :

```
var ombrePortee:DropShadowFilter=new DropShadowFilter(50, 45, 1, 0x000000,  
    ↳ 20, 10, 1, 3, false, false, false);  
galerie_mc.filters=[ombrePortee];
```



Définitions des paramètres du filtre ombre portée DropShadowFilter. Le filtre ombre portée se définit selon l'expression suivante :

```
dropShadowFilter:{distance:5, angle:45, alpha:1, color:0x000000,  
    ↳ blurX:10, blurY:10, strength:50, quality:3, type:"inner", knockout:false,  
    ↳ hideObject:false}
```

Les définitions des paramètres du filtre ombre portée sont :

- dropShadowFilter, le nom du filtre à appliquer reconnu comme tel par la classe filters.
- distance, indique le décalage en pixels de l'ombre par rapport à l'objet
- angle, désigne l'orientation de l'ombre par rapport à la source lumineuse. Une valeur de 45° génère une ombre portée légèrement décalée vers la droite. Tandis qu'une valeur négative l'oriente vers la gauche.
- alpha, indique l'opacité de l'ombre. La valeur s'exprime sur une échelle de 0 à 1, en valeur décimale. 0.5 indique une ombre semi-transparente.
- color, indique la couleur de l'ombre en hexadécimal. Cette valeur doit être précédée de 0x (zéro x). Par exemple, la valeur 0x000000 référence un noir. 0x désigne en ActionScript une valeur hexadécimale.
- blurX, indique l'étalement de l'ombre en pixels sur l'axe des X. Une valeur de 50 étale l'ombre sur une distance de 50 pixels en largeur.
- blurY, indique l'étalement de l'ombre en pixels sur l'axe des Y. Une valeur de 50 étale l'ombre sur une distance de 50 pixels en hauteur.
- strength, indique la force du filtre sur une échelle de 1 à 255. Ce paramètre diffuse l'effet.
- quality, indique la qualité de l'ombre. Cette valeur s'exprime sur une échelle de 1 à 3. 1 correspond à une ombre grossière mais moins gourmande en ressources graphiques. 2, à une ombre moyenne et 3, à une ombre fine, plus gourmande en ressources graphiques.
- type, désigne la manière dont l'ombre est appliquée à l'objet. Choisissez la valeur inner pour une ombre interne, la valeur outer pour une ombre externe et enfin, la valeur full pour une ombre mixte.
- knockout, valeur booléenne qui génère un masque à partir de l'effet et retire l'objet (forme un trou, un creux) en laissant visible l'arrière-plan.

Filtre halo

Le troisième filtre nommé halo crée et type un nouvel objet filtre halo GlowFilter(0xffffffff, 1, 20, 20, 3, 255, false, false) dont les valeurs désignent respectivement la couleur du halo, son alpha, son rayonnement flou en X et Y, sa force, sa qualité, son action sur l'intérieur ou non, et si l'objet halo masque le symbole ou non).

Le filtre est appliqué, plus tard dans la fonction, grâce à la commande suivante :

```
var halo:GlowFilter=new GlowFilter(0xffffffff, 1, 20, 20, 3, 255, false, false);
galerie_mc.filters=[halo];
```



Définitions des paramètres du filtre halo GlowFilter. Le filtre halo se définit selon l'expression suivante :

```
glowFilter:{color:0xffffffff, alpha:1, blurX:0, blurY:0, strength:1, quality:3,
type:"inner", knockout:false}
```

Les définitions des paramètres du filtre halo sont :

- glowFilter, le nom du filtre à appliquer reconnu comme tel par la classe filters.
- color, indique la couleur du halo en hexadécimal. Cette valeur doit être précédée de 0x (zéro x). Par exemple, 0xffffffff génère un halo blanc.
- alpha, indique l'opacité du halo. La valeur s'exprime sur une échelle de 0 à 1, en valeur décimale. La valeur 0.5 indique un halo semi transparent.
- blurX, indique l'étalement du halo en pixels sur l'axe des X. Une valeur de 50 étale le halo sur une distance de 50 pixels en largeur.
- blurY, indique l'étalement du halo en pixels sur l'axe des Y. Une valeur de 50 étale le halo sur une distance de 50 pixels en hauteur.
- strength, indique la force du filtre sur une échelle de 1 à 255. Ce paramètre diffuse l'effet.
- quality, indique la qualité du halo. Cette valeur s'exprime sur une échelle de 1 à 3. 1 correspond à un halo grossier mais moins gourmand en ressources graphiques. 2, à un halo moyen et 3, à un halo fin, plus gourmand en ressources graphiques.
- type, désigne la manière dont le halo est appliqué à l'objet. Choisissez la valeur inner pour un halo interne, la valeur outer pour un halo externe et enfin, la valeur full pour un halo mixte.
- knockout, valeur booléenne qui génère un masque à partir de l'effet et masque ou retire l'objet en laissant visible l'arrière-plan.

Filtre biseau

Enfin, un quatrième filtre nommé biseau crée et type un nouvel objet filtre biseau BevelFilter(50,45,0xFFFFFFFF,1,0x000000,1,50,50,100,3,"inner",false) dont les paramètres désignent respectivement la distance, l'angle, la couleur du biseau exposée à la lumière, son alpha, la couleur de l'ombre du biseau, l'alpha de l'ombre, son flou en X et Y, sa force, sa qualité, si l'effet est intérieur (inner), extérieur (outer), ou mixte (full), et si le masque est activé ou non.

Le filtre est appliqué, plus tard dans la fonction, grâce à la commande suivante :

```
var biseau:BevelFilter=new BevelFilter(50,45,0xFFFFFFFF,1,0x000000,1,50,
50,100,3,"inner",false);
galerie_mc.filters=[biseau];
```



Définitions des paramètres du filtre biseau BevelFilter. Le filtre biseau se définit selon l'expression suivante :

```
bevelFilter:{distance:5, angle:45, highlightColor:0xffffffff, highlightAlpha:1,
shadowColor:0x000000, shadowAlpha:1, blurX:0, blurY:0, strength:1, quality:3,
type:"inner", knockout:false}
```

Les définitions des paramètres du filtre biseau sont :

- `bevelFilter`, le nom du filtre à appliquer reconnu comme tel par la classe `filters`.
- `distance`, indique le décalage en pixels du biseau par rapport à l'objet.
- `angle`, désigne l'orientation du biseau par rapport à la source lumineuse. Une valeur de 45° génère un biseau légèrement décalé vers la droite. Tandis qu'une valeur négative l'oriente vers la gauche.
- `highlightColor`, indique la valeur de couleur du biseau qui reçoit la lumière. Généralement situé en haut et à gauche, il peut être inversé selon la valeur définie pour l'angle.
- `highlightAlpha`, indique l'opacité de la couleur du biseau exposé à la lumière. Agit sur le paramètre précédent `highlightColor`.
- `shadowColor`, indique la valeur de couleur du biseau qui reçoit l'ombre. Généralement situé en bas et à droite, il peut être inversé selon la valeur définie pour l'angle.
- `shadowAlpha`, indique l'opacité de la couleur du biseau ombré. Agit sur le paramètre précédent `shadowColor`.
- `blurX`, indique l'étirement du biseau en pixels sur l'axe des X. Une valeur de 50 étire le biseau sur une distance de 50 pixels en largeur.
- `blurY`, indique l'étirement du biseau en pixels sur l'axe des Y. Une valeur de 50 étire le biseau sur une distance de 50 pixels en hauteur.
- `strength`, indique la force du biseau sur une échelle de 1 à 255. Ce paramètre diffuse l'effet et adoucit le biseau.
- `quality`, indique la qualité du biseau. Cette valeur s'exprime sur une échelle de 1 à 3. 1 correspond à un biseau grossier mais moins gourmand en ressources graphiques. 2, à un biseau moyen et 3, à un biseau fin, plus gourmand en ressources graphiques.
- `type`, désigne la manière dont le biseau est appliqué à l'objet. Choisissez la valeur `inner` pour un biseau interne, la valeur `outer` pour un biseau externe et enfin, la valeur `full` pour un biseau mixte ou estampé.
- `knockout`, valeur booléenne qui génère un masque à partir de l'effet et masque ou retire l'objet en laissant visible l'arrière-plan.

Déclinaison des filtres

Dans notre exemple, une fois les filtres définis nous initialisons certaines variables :

```
var i:Number=1;
var nombreDePhotos:Number=5;
var dureeBoucle:Number=10000;
var boucle:Timer=new Timer(dureeBoucle,nombreDePhotos);
```

Nous utilisons les mêmes variables que dans l'exemple de la section précédente pour définir les valeurs attendues par le chronomètre `Timer`. Reportez-vous à la section précédente pour plus de détails sur ces valeurs. Puis, nous masquons la galerie par défaut :

```
galerie_mc.visible=false;
```

Plus loin, comme dans l'exemple précédent, une fonction génère le défilement des images au rythme des itérations imposées par le chronomètre. Cette fonction est également exécutée :

une fois dès la lecture du document Flash avec `lancerGalerie()` ; et à chaque itération par le chronomètre, avec la fonction `jouerBoucle(evt:TimerEvent)`.

```
//----- actions

boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
boucle.start();

function lancerBoucle(evt:TimerEvent) {
    lancerGalerie();
}

function lancerGalerie() {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        //INITIALISATION
        //(../...)
        //ANIMATION INTRO
        //(../...)
        //ANIMATION SORTIE
        //(../...)
        i++;
    } else {
        galerie_mc.visible=false;
    }
}
lancerGalerie();
```

Dans la fonction, nous observons trois parties. L'initialisation, l'animation de départ et l'animation en sortie. L'animation d'un filtre est distribuée dans chacune de ces étapes. Prenons pour commencer l'exemple du filtre flou, comme suit :

```
// INITIALISATION
galerie_mc.filters=[frou];
//ANIMATION INTRO
TweenMax.from(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100, quality:3},
delay:0, ease:Strong.easeInOut});
//ANIMATION SORTIE
TweenMax.to(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100, quality:3},
delay:5, ease:Strong.easeInOut});
```

L'animation que nous programmons fonctionne de la manière suivante : d'abord, la galerie est nette une fraction de seconde, mais cela est imperceptible : nous initialisons le filtre flou (partie initialisation du code). Ensuite, et instantanément, la galerie apparaît floue, puis, progressivement, l'image devient nette et légèrement bombée avec un biseau très diffus (partie animation intro du code). Puis, après une pause d'une seconde, progressivement l'image redevient entièrement floue (partie animation sortie du code). C'est seulement, une fois cette seconde boucle terminée que l'image suivante est affichée, le filtre initialisé, et que l'animation reprend et ainsi de suite, de boucle en boucle, jusqu'à la dernière image de la galerie.

En publiant le document, nous observons que plusieurs filtres sont appliqués simultanément. Vous pouvez supprimer les lignes de commande de chaque filtre individuellement pour mieux en apprécier les effets distinctifs si vous le souhaitez.

Dans le cheminement de notre animation, il est indispensable d'initialiser d'abord les valeurs du filtre que nous animons ensuite, pour la simple raison que notre animation se termine par un flou général. Si nous n'initialisons pas les valeurs, l'animation suivante partira d'une image déjà floue pour y ajouter du flou de même intensité. Nous aurions donc l'impression que l'animation n'agit pas et que l'image reste floue tout le temps. En initialisant les valeurs par l'application du filtre dès l'exécution de la fonction, nous assurons le contrôle du rendu de notre effet en gardant actives ses valeurs.

Pour animer chaque filtre, dans notre exemple, nous commençons par initialiser les valeurs en appliquant directement chaque filtre que nous avons prédéfini en amont, sur l'objet `galerie_mc`, puis, nous utilisons la classe `Greensock TweenMax` pour animer le filtre en entrée et en sortie :

```
// INITIALISATION
galerie_mc.filters=[flou];
//ANIMATION INTRO
TweenMax.from(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100, quality:3},
➤ delay:0, ease:Strong.easeInOut});
//ANIMATION SORTIE
TweenMax.to(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100, quality:3},
➤ delay:5, ease:Strong.easeInOut});
```

Observons que le filtre est introduit dans une interpolation `TweenMax` comme n'importe quelle autre propriété, entre deux virgules. Le filtre est contenu entre deux parenthèses et l'ensemble des paramètres du filtre est contenu entre deux accolades. Chaque propriété du filtre est séparée par une virgule.



Faut-il renseigner toutes les propriétés pour les filtres ? Lorsqu'un filtre est placé en paramètre d'une interpolation de type `TweenMax`, il n'est pas nécessaire de spécifier toutes les propriétés disponibles pour le filtre si celles-ci ne requièrent pas d'être modifiées. Leur ordre d'apparition, de même, n'a pas d'incidence, pourvu que leur dénomination soit précisée. Si aucune dénomination n'est spécifiée, l'ordre alors est important et correspond, pour chaque filtre individuellement, à celui que nous avons présenté dans les notes de cette section. Dans notre exemple, nous spécifions volontairement toutes les propriétés possibles pour l'animation d'introduction, afin de vous permettre d'en prendre connaissance, mais nous ne les répétons pas pour l'animation de sortie, car les valeurs de ces propriétés ne changent pas et les valeurs par défaut, si elles n'ont pas été modifiées entre temps, s'appliquent automatiquement.

Lorsqu'un filtre est appliqué directement, indépendamment de la classe `TweenMax`, à l'initialisation par exemple, le typage n'est pas indiqué, et tous les paramètres ne sont pas requis, mais leur ordre est important et reprend celui également renseigné dans les notes de ce chapitre, dans chaque définition de filtre que nous avons développée.

Nous savons à présent que la classe `TweenMax` permet de temporiser l'animation grâce au paramètre `delay` (voir Chapitre 2). Nous n'avons donc pas besoin de détecter la fin de l'animation avec la classe `TweenEvent.MOTION_FINISH` comme nous l'aurions fait dans un autre contexte, pour enchaîner les animations. Il suffit ici de décaler les animations dans le temps avec le paramètre `delay` et nous les verrons se succéder les unes à la suite des autres. Veillons toutefois à ne pas croiser les animations dans le temps. Si la première animation

dure 4 secondes, la deuxième série d'animations démarrera évidemment à plus de 4 secondes, par sécurité (delay:5).

Les paramètres from et to indiquent respectivement de démarrer les interpolations soit en partant (from) soit en terminant (to) sur les valeurs indiquées en paramètre de l'interpolation TweenMax, depuis les valeurs courantes des propriétés appliquées à l'objet (from), ou vers ces propriétés (to). Les propriétés courantes de l'objet galerie_mc correspondent à celles appliquées par l'initialisation du filtre à l'étape qui précède la première animation d'introduction.

Nous déclinons le principe sur les quatre filtres et obtenons :

```
function lancerGalerie() {
    galerie_mc.visible=true;
    if (i<=nombreDePhotos) {
        galerie_mc.gotoAndStop(i);
        // INITIALISATION
        galerie_mc.filters=[flou];
        galerie_mc.filters=[ombrePortee];
        galerie_mc.filters=[halo];
        galerie_mc.filters=[biseau];
        //ANIMATION INTRO
        TweenMax.from(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100,
            quality:3}, delay:0, ease:Strong.easeInOut});
        TweenMax.from(galerie_mc, 4, {dropShadowFilter:{distance:5, angle:45,
            ➤ alpha:1, color:0x000000, blurX:10, blurY:10, strength:1,
            ➤ quality:3, type:"inner", knockout:false, hideObject:false},
            ➤ delay:0, ease:Strong.easeInOut});
        TweenMax.from(galerie_mc, 4, {glowFilter:{color:0xffffffff, alpha:1,
            ➤ blurX:0, blurY:0, strength:1, quality:3, type:"inner",
            ➤ knockout:false}, delay:0, ease:Strong.easeInOut});
        TweenMax.from(galerie_mc, 4, {bevelFilter:{distance:5, angle:45,
            ➤ highlightColor:0xffffffff, highlightAlpha:1,
            ➤ shadowColor:0x000000, shadowAlpha:1, blurX:0, blurY:0,
            ➤ strength:1, quality:3, type:"inner", knockout:false},
            ➤ delay:0, ease:Strong.easeInOut});
        //ANIMATION SORTIE
        TweenMax.to(galerie_mc, 4, {blurFilter:{blurX:100, blurY:100, quality:3},
            ➤ delay:5, ease:Strong.easeInOut});
        TweenMax.to(galerie_mc, 4, {dropShadowFilter:{distance:5, angle:45,
            ➤ alpha:1, color:0x000000, blurX:10, blurY:10}, delay:5,
            ➤ ease:Strong.easeInOut});
        TweenMax.to(galerie_mc, 4, {glowFilter:{color:0xffffffff, alpha:1, blurX:0,
            ➤ blurY:0}, delay:5, ease:Strong.easeInOut});
        TweenMax.to(galerie_mc, 4, {bevelFilter:{distance:5, angle:45,
            ➤ color:0x000000, blurX:0, blurY:0}, delay:5, ease:Strong.
            ➤ easeInOut});
        i++;
    } else {
        galerie_mc.visible=false;
    }
}
```

À retenir

- La classe `filters` est native et permet d'ajouter des filtres directement sur des objets de la scène.
- La classe `filters` peut être enchaînée avec d'autres animations grâce à l'utilisation du paramètre `delay` de la classe `Greensock TweenMax`, voire grâce à la création d'un chronomètre `Timer`.
- Il est important d'initialiser les paramètres au début de chaque animation si l'on veut garantir l'effet boucle d'une animation dont les effets affectent les contenus à la fois en entrée et en sortie.
- L'ordre d'affichage des propriétés dans un filtre n'a pas d'incidence si leur désignation est spécifiée.
- Toutes les propriétés d'un filtre ne requièrent pas d'être renseignées si leurs valeurs ne changent pas.

Synthèse

Vous avez appris à appliquer des effets et des filtres à des objets de la scène et à programmer des interpolations d'effets et de filtres à travers des animations générées dynamiquement dans le temps. En plus de la classe `TweenEvent.MOTION_FINISH`, vous avez également appris à enchaîner des animations sur l'échelle du temps grâce à l'utilisation du chronomètre `Timer`, de sa sous-classe `TIMER_EVENT.TIMER_COMPLETE`, des sous-classes `allTransitionsInDone`, `allTransitionsOutDone` de la classe `transition`, et du paramètre `delay` de la classe `Greensock TweenMax`. Vous êtes à présent en mesure de planifier des actions dans le temps en les couplant à des effets graphiques animés et avancés.

4

La programmation de squelettes

Introduction

Un squelette est une structure hiérarchique qui relie, avec un lien de parenté, différents symboles de la scène ou différentes parties d'une même forme graphique continue. En déployant l'ossature du squelette, les symboles ou les parties de la forme graphique suivent la progression du mouvement de la structure qui le compose.

Pour en programmer l'animation, nous utilisons la classe *ik* (*inverse kinematic* en anglais, pour cinématique inverse). Cette classe, à ce jour, ne permet pas de réaliser directement un squelette en programmation, mais uniquement, de l'animer. Les exemples proposés possèdent donc déjà un squelette, mais nous aborderons également la manière de construire cet objet manuellement pour la programmation.

Une structure articulée autour d'un squelette permet, entre autre, de reconstituer l'ossature d'un personnage et de l'animer. Elle permet aussi de distribuer les contenus d'un site sous la forme de dispositifs hiérarchisés, comme un menu ou une arborescence. Mais, si cette armature est associée à une forme graphique, cela nous donne aussi accès à des interpolations de forme accessibles en programmation.

Dans ce chapitre, nous réalisons dans un premier temps une animation mécanique d'un droïde dont les mouvements réagissent à la position du pointeur. Nous abordons ensuite l'animation de squelettes de formes organiques avec des formes graphiques vectorielles, pour animer, par exemple, le mouvement d'un végétal selon la position d'un objet animé (une abeille). Nous présentons ensuite comment gérer une animation programmée en mode interactif, afin que l'utilisateur puisse lui-même modifier le positionnement du squelette en déplaçant chaque segment de la structure manuellement.

Les squelettes, lorsqu'ils sont déployés dans des SWF imbriqués, ne s'exécutent plus. Nous présentons donc aussi une solution pour l'importation de squelettes en mode Exécution, à l'intérieur de documents Flash imbriqués.

Programmer un mouvement mécanique

L'animation de squelettes à partir de symboles permet de mettre en mouvement des structures mécaniques qui font s'articuler des objets entre eux, et où aucun objet ne se mélange jamais aux autres. Ce type de structure à base de symboles convient à l'animation de robots, de cycles de marche mécaniques, de grues ou de systèmes de navigation, entre autres.

Dans cet exemple, un squelette est déjà en place dans le scénario. Il représente le corps d'un droïde et se compose de 15 symboles chacun relié à l'autre par un segment, *IKBone* (voir Figure 4.1). Chaque segment possède deux extrémités de jointure ou liaisons (*IKJoint*) placées respectivement en tête (*headJoint*) et en queue (*tailJoint*). C'est à partir de ces

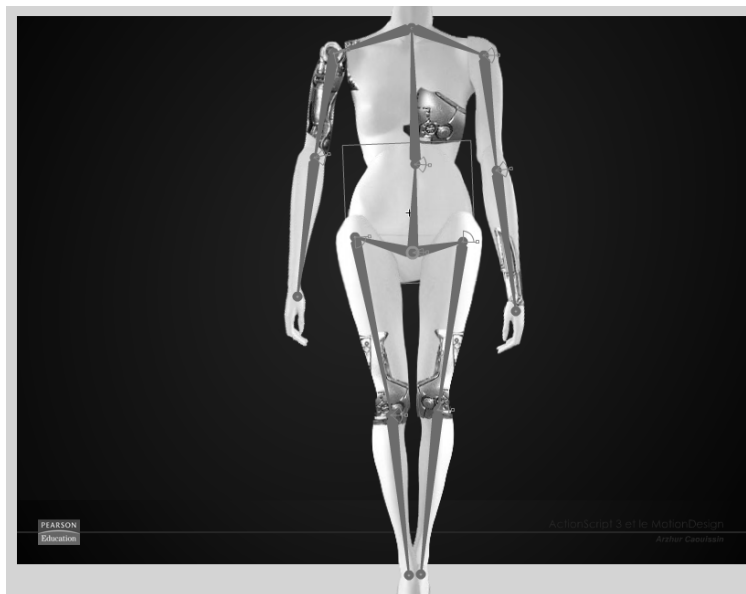
axes de rotation et de placement que nous déterminons les mouvements à travers des interpolations cinématiques (IKMover). La spécificité de ces interpolations est de répercuter sur chaque symbole faisant partie de la hiérarchie, de nouvelles valeurs de positionnement, selon les contraintes de mouvement définies dans l'ossature lors de sa construction. L'ensemble du squelette est, quant à lui, inclus dans une armature générale (IKArmature) dont on peut déterminer le mode d'affichage avec une méthode (IKManager) et l'afficher, soit pour l'animation, soit pour l'exécution.



Heure de création et exécution. La terminologie des modes d'affichage des squelettes peut surprendre. Nous observons en effet, dans l'Inspecteur de propriétés, les deux options *Heure de création* et *Exécution* littéralement traduites des termes anglais *AuthorTime* et *RunTime*. Comprenez, en ces termes, *Gestion de l'animation dans l'interface auteur* (*AuthorTime*, ou *Heure de création*) d'une part et *Gestion de l'animation à la publication* (*RunTime*, ou *Exécution*) d'autre part. Nous devrions ainsi plutôt parler de l'option "Animation" pour le premier et d'"Exécution" pour le second.

Figure 4.1

Aperçu
du document.



Exemples > ch4_ProgrammationDeSquelettes_1.fla

Le document que nous utilisons présente la structure suivante : dans la scène, au-dessus du calque *fond_mc*, apparaît un calque nommé *squelette_Droide* (voir Figure 4.2). Chaque symbole possède un nom d'occurrence qui définit clairement la partie du corps à laquelle il se rattache et le désigne en tant qu'objet structurel d'un squelette (*ikNode_bassin*, *ikNode_torax*, *ikNode_cou*, *ikNode_brasD*, *ikNode_avantBrasD*, *ikNode_mainD*, etc.). De même, chaque segment qui relie ces symboles entre eux possède son propre nom d'occurrence qui identifie la partie de l'ossature à laquelle il est rattaché et le désigne en tant qu'objet structurel de liaison (*ikBone_colonneBas*, *ikBone_colonneHaut*, *ikBone_epauleD*,

ikBone_humerusD, ikBone_cubitusD, etc.). À travers ces identifiants, nous distinguons bien les symboles (*Nodes*) de leurs jonctions (*Bones*). Le squelette, lui, possède également son propre nom d'occurrence *Squelette_Droide*.

Figure 4.2

Aperçu du scénario de la scène principale.



Dans le calque nommé *actions*, nous pouvons lire le code suivant :

```
//----- initialisation
import fl.ik.*;

//----- actions
// définition du squelette

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Droide");
➔ squelette.registerElements(stage);

var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0);
var segment_colonneHaut:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0);
var segment_epauleD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➔ getChildAt(0);
var segment_humerusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➔ getChildAt(0).getChildAt(0);
var segment_cubitusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➔ getChildAt(0).getChildAt(0).getChildAt(0);
var segment_epauleG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➔ getChildAt(1);
var segment_humerusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➔ getChildAt(1).getChildAt(0);
var segment_cubitusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➔ getChildAt(1).getChildAt(0).getChildAt(0);
var segment_illiaqueG:IKJoint=squelette.rootJoint.getChildAt(1);
var segment_femurG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0);
var segment_tibiaG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0).
➔ getChildAt(0);
var segment_illiaqueD:IKJoint=squelette.rootJoint.getChildAt(2);
var segment_femurD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0);
var segment_tibiaD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0).
➔ getChildAt(0);

trace(squelette.rootJoint.getChildAt(0).getChildAt(0).name)

// animation du squelette

var mouvement1:IKMover= new
➔ IKMover(segment_humerusD,segment_humerusD.position);
```



```
stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
    var arrivee:Point=new Point(mouseX,mouseY);
    mouvement1.moveTo(arrivee);
}
```

Construire un squelette pour la programmation

Avant de programmer l'animation d'un squelette, il convient de s'assurer qu'il est préalablement bien structuré. Tout d'abord, nous utilisons la classe `ik` qui permet de déplacer les têtes de chaque segment. Il faut donc bien comprendre où se positionnent ces têtes et comment les placer judicieusement pour garantir la cohérence de l'animation. Pour construire une armature pour la programmation, procédez comme suit :

1. Placez un certain nombre de symboles de type `MovieClip` sur un même calque.
2. Repositionnez éventuellement les centres de transformation de chaque symbole (grâce au petit rond blanc que l'on peut déplacer avec l'outil de transformation libre). C'est en effet sur ces centres que les segments seront magnétisés et c'est leur emplacement qui détermine la cohérence de l'animation. Pour visualiser et déplacer un centre de transformation, utilisez l'outil de transformation ou activez le raccourci `Q`, puis glissez-déposez le petit cercle blanc à l'emplacement souhaité. Puis, réactivez l'outil de sélection en appuyant sur la touche `V`. Pour faciliter le positionnement des centres, pensez à désactiver au besoin les options d'accrochage du menu `Affichage`.
3. Si les symboles qui composent votre sujet à animer reposent sur des images bitmap (PSD, PNG, JPEG, Gif), activez le lissage des bitmaps pour éviter l'apparition du crénelage lorsque les symboles pivoteront (voir Chapitre 11).
4. Placez les segments sur les symboles. Dans la barre d'outils, utilisez l'outil `Segment`, en forme d'os, ou le raccourci `X`. Puis, cliquez sur le centre de transformation du symbole père, glissez et relâchez le bouton de la souris sur le centre de transformation du symbole enfant.
5. Reproduisez la manipulation autant de fois que vous possédez de symboles à relier entre eux, en partant toujours de l'extrémité située en queue d'un segment, puis poursuivez la hiérarchie vers les autres symboles.
6. Ajoutez un nom d'occurrence pour chaque objet créé. Pour cela, cliquez sur les segments un à un, sur les têtes de segments une à une, et sur le calque du squelette lui-même, pour leur attribuer à tous des noms d'occurrence distinctifs. Flash en propose par défaut (`IkNode_1` pour les têtes de segment, `IkBoneName1` pour les segments et `Squelette_1` pour le calque du squelette). Vous pouvez conserver ces noms ou les adapter à votre perception.
7. Activez ou non les contraintes de mouvement et de rotation. Pour ce faire, sélectionnez chaque segment avec l'outil de sélection (flèche noire ou raccourci `V`), et depuis l'Inspecteur de propriétés, activez les différentes options de liaison (rotation, translationX et translationY). Attention, les valeurs indiquent une rotation relative. Pour faciliter les manipulations, orientez chaque symbole sur l'axe médian de l'espace de rotation que vous souhaitez lui réserver. En activant l'option de rotation, Flash distribue automatiquement une zone de 45° (paramétrable) de part et d'autre de la position courante de l'objet.
8. Le squelette doit être activé pour le mode Exécution. Pour cela, cliquez sur le calque du squelette. Puis, dans l'Inspecteur de propriétés, dans le menu `Type` de la catégorie `Options`, sélectionnez `Exécution`.

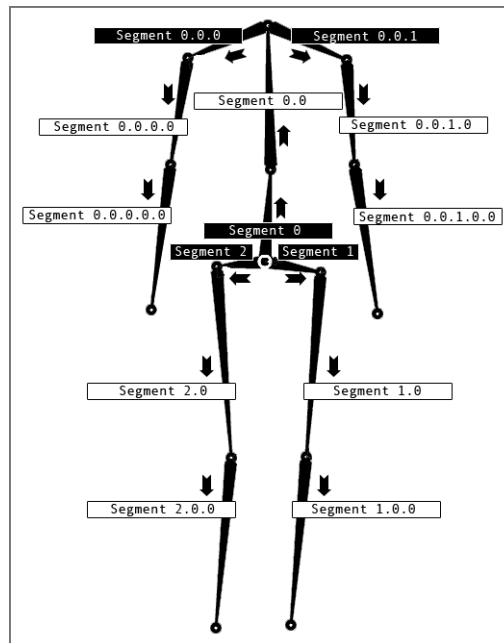
Définition du squelette

Pour comprendre la manière dont nous interagissons en ActionScript avec les éléments du squelette, nous devons au préalable bien saisir la manière dont celui-ci a été construit. Dans notre exemple, nous disposons d'une armature composée de 15 segments. Certains d'entre eux sont rattachés au même nœud de liaison. Chaque branche qui se divise, dans la hiérarchie descendante de l'armature, n'est pas considérée comme un ensemble indépendant, mais bien comme faisant partie d'un même ensemble. Ce qui distingue une division par rapport à une autre est son ordre d'apparition dans la liste d'affichage du squelette (ce qui équivaut par défaut à l'ordre de création des nœuds), ou à son nom d'occurrence. Pour contrôler une branche plutôt qu'une autre, nous ciblerons donc celle-ci par son ordre d'affichage (`getChildAt(0)`) ou par son nom d'occurrence (`getNodeByName("nomDuNoeud")`).

Ainsi, avant de programmer des actions, il est souhaitable d'identifier clairement la hiérarchie dont nous disposons. Pour notre exemple, nous avons réalisé un schéma qui reprend l'ordre de chaque segment de liaison tel qu'identifié depuis la liste d'affichage (voir Figure 4.3) (nous présentons la manière d'obtenir l'ordre d'affichage de chaque segment dans la lecture du code qui suit).

Figure 4.3

Aperçu
de la structure
du squelette.



Dans notre exemple à nouveau, relevons que nous avons commencé la construction du squelette à partir du bassin, en progressant d'abord vers le cou, puis la jambe gauche et enfin vers la jambe droite. Le bassin se divise donc, dès le départ, en trois branches distinctes que nous pouvons identifier par la branche 0 (cou), la branche 1 (jambe gauche) et la branche 2 (jambe droite). Le cou, à son tour, se divise en deux parties, une pour chaque bras.

Nous obtenons donc les subdivisions suivantes : avec la branche 0 qui correspond au bras droit, et 1 qui correspond au bras gauche. Mais, dans le contexte hiérarchique qui part du bassin, et se prolonge par le thorax avant d'atteindre les épaules, nous pouvons identifier la branche du bras droit comme liaison 0.0.0 et celle du bras gauche comme liaison 0.0.1. Et ainsi de suite. En somme, à chaque progression vers un sous-niveau de l'armature, nous ajoutons une valeur.



Création de squelettes humains. Le livre *L'art du bluff avec Flash CS4*, de Chris Georgennes, aux éditions Pearson, propose d'ajouter un segment à l'aplomb du bassin afin de le relier au sol. Pour de plus amples renseignements sur la construction de squelettes d'animation, notamment humains, reportez-vous à cet ouvrage.

Le code affiché est composé de trois parties. D'abord, nous définissons les classes et les variables à utiliser (initialisation). Ensuite, nous structurons les éléments destinés à être animés (identification des éléments animables). Enfin, nous réalisons l'animation. Celle-ci se caractérise d'abord par le contrôle du mouvement des liaisons du squelette (animation du squelette), puis par la mise en relation de ces mouvements avec le pointeur à travers la fonction.

Nous commençons par importer les classes requises pour l'animation du squelette :

```
//----- initialisation
import fl.ik.*;
```

La classe `ik` est utilisée pour la gestion des mouvements de liaison du squelette. L'astérisque sous-entend l'importation des sous-classes `IKArmature` (définition du squelette), `IKBone` (définition des segments qui contiennent les liaisons), `IKEvent` (définition des événements pour les fonctions), `IKJoint` (définition des liaisons à animer), `IKManager` (définition du statut du squelette) et `IKMover` (définition des animations en tant que telles).

Puis, nous activons le squelette pour pouvoir, ensuite, en animer la composante :

```
//----- actions
// définition du squelette

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Droide");
squelette.registerElements(stage);
```

Dans ces premières lignes, grâce à la sous-classe `IKManager` qui permet de statuer sur le comportement du squelette, nous affectons le contrôle du squelette à la scène courante. Mais cette instruction est optionnelle car l'API de Flash le gère automatiquement :

```
IKManager.setStage(stage);
```

Puis, nous convertissons le squelette de la scène en objet `IKArmature` programmable. Pour cela, nousinstancions un nouvel objet `IKArmature` en reprenant, en paramètre de la méthode `getArmatureByName()`, le nom d'occurrence du squelette que nous avons pu identifier depuis l'Inspecteur de propriétés directement sur la scène principale :

```
var squelette:IKArmature= IKManager.getArmatureByName("Squelette_Droide");
```

Pour cibler le squelette, nous utilisons ici `getArmatureByName` puisque nous en connaissons le nom. Mais nous aurions aussi bien pu le cibler par son ordre d'affichage, dans la liste d'affichage des squelettes indépendante de la liste d'affichage de la scène, avec la méthode `getArmatureAt()`.

Nous mémorisons ensuite toute l'ossature du squelette présente sur la scène de sorte à en autoriser le contrôle avec la méthode `registerElements()` :

```
squelette.registerElements(stage);
```

Une fois que nous avons instancié la structure globale, nous pouvons à présent entrer dans les maillons du squelette et déclarer chacune des liaisons dont nous souhaitons pouvoir contrôler la position :

```
var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0);
```

Pour déclarer la première liaison, nous créons un nouvel objet `IKJoint`. Nous ciblons le premier nœud de liaison en reprenant d'abord le nom du squelette véhiculé par la variable `squelette`, puis, ciblons le nœud racine avec `rootJoint`, c'est-à-dire le centre de transformation du bassin bas, puis, la première branche qui remonte la colonne vertébrale avec `getChildAt(0)` (voir Figure 4.4).

Pour vérifier que l'ordre d'apparition choisi en paramètre correspond bien à l'objet que nous souhaitons rendre disponible sur la scène, nous utilisons une action `trace` en reprenant le ciblage auquel nous associons la propriété `.name` pour connaître le nom d'occurrence de l'objet :

```
trace(squelette.rootJoint.getChildAt(0).getChildAt(0).name)
```

Nous obtenons, dans la fenêtre de sortie, la réponse suivante :

```
Segment_cou
```

Nous reproduisons ensuite le principe pour tous les segments, afin d'en permettre l'animation :

```
var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0);
var segment_colonneHaut:IKJoint=squelette.rootJoint.getChildAt(0).
➤ getChildAt(0);
var segment_epauleD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➤ getChildAt(0);
var segment_humerusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➤ getChildAt(0).getChildAt(0);
var segment_cubitusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➤ getChildAt(0).getChildAt(0).getChildAt(0);
var segment_epauleG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➤ getChildAt(1);
var segment_humerusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➤ getChildAt(1).getChildAt(0);
var segment_cubitusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➤ getChildAt(1).getChildAt(0).getChildAt(0);
var segment_illiaqueG:IKJoint=squelette.rootJoint.getChildAt(1);
var segment_femurG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0);
var segment_tibiaG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0).
➤ getChildAt(0);
var segment_illiaqueD:IKJoint=squelette.rootJoint.getChildAt(2);
var segment_femurD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0);
var segment_tibiaD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0)
➤ getChildAt(0);
```

Observons que pour atteindre un maillon enfant, nous utilisons la technique de ciblage pointé en ajoutant, pour chaque nouveau segment enfant, la méthode `getChildAt()` avec le

numéro qui correspond à l'ordre d'affichage de l'enfant ciblé, conformément au schéma observé plus haut.



Il est possible de cibler uniquement un segment à partir de son identifiant, comme suit :

```
var segment1:IKBone=squelette.getBoneByName("ikBoneName1");
```

Ou à partir de son ordre d'affichage :

```
var segment1:IKBone=squelette.getBoneByName("ikBoneName1");
```

Une fois les liaisons définies, il ne reste plus qu'à animer.

Animer le squelette

La propriété rotation n'étant pas autorisée en écriture par la classe IK, pour animer une liaison, nous définissons donc un nouvel objet de transition qui déplace une liaison d'un point donné en X et Y à un autre point en X et Y. Pour mieux comprendre le procédé, simplifions notre exemple en nous concentrant d'abord sur le fonctionnement d'une animation de liaison :

```
// animation du squelette
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
var arrivee:Point=new Point(0,0);
mouvement1.moveTo(arrivee);
```

Nous déclarons ici, en premier, l'objet IKMover qui constitue le moteur de l'animation et redistribue, pour chaque segment lié à la chaîne courante, les valeurs nécessaires à leur éventuel repositionnement :

```
var mouvement1:IKMover= new IKMover(segment_humerusD,segment_humerusD.position);
```

L'animation appelle deux paramètres qui sont respectivement l'objet à animer (en l'occurrence, la liaison segment_humerusD définie précédemment), et les coordonnées du point de départ de l'animation de cet objet, à partir d'une valeur de type Point(X,Y).



Définition du constructeur Point(). Une valeur de type Point(X,Y) véhicule, dans une seule variable, les propriétés x et y de l'objet auquel elle est rattachée. Ce constructeur peut, pour information, être appelé par les classes suivantes : BitmapData, DisplayObject, DisplayObjectContainer, DisplacementMapFilter, NativeWindow, Matrix ou Rectangle.

Ainsi, nous pouvons déterminer la valeur de position de départ à travers une nouvelle variable ou en reprenant la position courante de l'objet à animer. Avec une variable, nous obtenons :

```
var depart:Point=new Point(100,50) ;
var mouvement1:IKMover= new IKMover(segment_humerusD,depart);
```

En reprenant directement la propriété position de l'objet courant, nous obtenons :

```
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
```

Nous récupérons directement la position courante de l'objet à animer. Puis, nous définissons le point d'arrivée :

```
var arrivee:Point=new Point(0,0);
```

Et enfin, nous activons l'animation :

```
mouvement1.moveTo(arrivee);
```

L'animation reprend l'objet IKMover initié plus haut et utilise la méthode moveTo afin de créer une interpolation entre le point de départ défini dans l'objet IKMover et le point d'arrivée passé ici en paramètre.

Dans le fichier exemple de cette section, l'interpolation que nous venons de décrire, est placée dans une fonction associée à un gestionnaire d'événements de type Event.ENTER_FRAME afin que le calcul de la position puisse être redéfini perpétuellement. Nous plaçons, en paramètre du point d'arrivée, les valeurs qui correspondent à la position courante du pointeur avec mouseX et mouseY. Nous obtenons :

```
// animation du squelette
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
    var arrivee:Point=new Point(mouseX,mouseY);
    mouvement1.moveTo(arrivee);
}
```

Pour optimiser encore plus l'animation, vous pouvez éventuellement typer la variable arrivee en dehors de la fonction, et la mettre à jour seulement dans la fonction, comme suit :

```
// animation du squelette
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
var arrivee:Point ;
stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
    arrivee=new Point(mouseX,mouseY);
    mouvement1.moveTo(arrivee);
}
```

En modifiant le paramètre qui appelle le nom de l'objet à animer, dans l'objet IKMover, vous pouvez animer le squelette à partir de chacun des axes de rotation (voir Figures 4.4 et 4.5).

À retenir

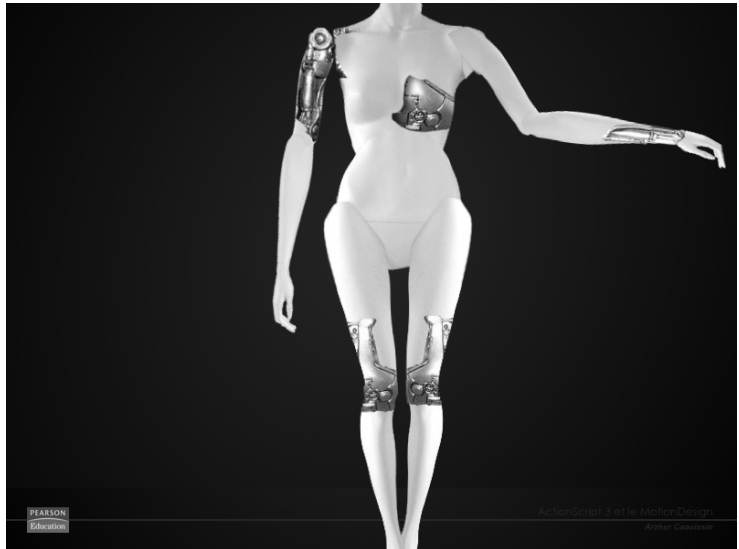
- La classe ik permet d'animer la position des liaisons établies entre plusieurs segments d'un même squelette.
- Pour programmer l'animation d'un squelette, vous devez au préalable construire le squelette dans l'interface auteur de Flash en activant l'affichage pour l'exécution depuis l'Inspecteur de propriétés.
- Pour faciliter la gestion de l'articulation des segments, vous devez placer judicieusement les centres de transformation de chaque symbole avant la création du squelette.
- Pour faciliter la définition des liaisons à animer, vous devez clairement identifier l'ordre d'affichage de ces éléments.
- Il est possible d'associer le mouvement d'une liaison à la position courante du pointeur.

Figure 4.4

Aperçu de l'animation avec l'objet `segment_tibiaD`.

**Figure 4.5**

Aperçu de l'animation avec l'objet `segment_avant-BrasG`.



Programmer un mouvement organique

De la même manière que nous animons des squelettes de symboles, nous pouvons aussi programmer l'animation de squelettes pour des formes graphiques vectorielles.

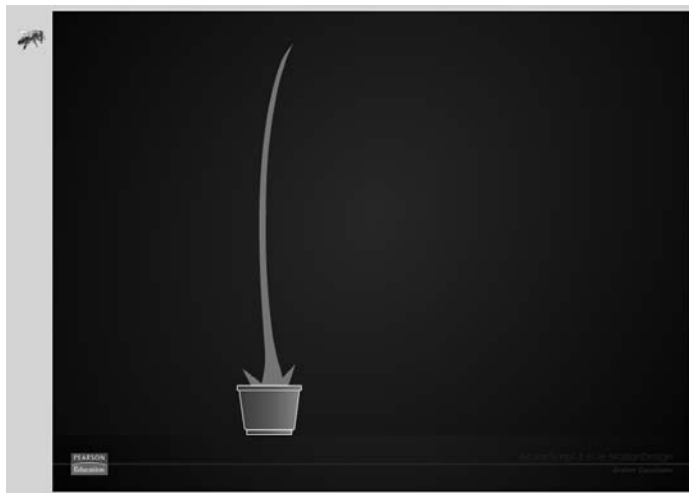
L'utilisation des squelettes avec les formes vectorielles permet de réaliser, par exemple, des animations de "lipsync" (mouvement de bouche), des animations végétales, organiques, liquides ou hétérées (radiosité, vapeurs, nuages), entre autres, mais pas de formes transparentes.

Dans cette section, nous utilisons l'animation d'une plante qui se courbe au passage d'une abeille. Nous utilisons ici les paramètres de la méthode `moveTo` pour magnétiser la dernière

liaison de la plante sur l'abscisse du symbole abeille (voir Figure 4.6). L'abeille, elle, est animée à l'aide de la classe Greensock TweenMax.

Figure 4.6

Aperçu du document.



Exemples > ch4_programmationDeSquelettes_2.fla

Dans la scène de notre document, nous pouvons voir, au-dessus du calque `fond_mc`, un squelette attaché à une forme graphique nommé `Squelette_Herbe`. Au premier plan de cette armature, apparaissent quelques feuilles et un pot pour la décoration, puis un symbole de type MovieClip nommé `abeille_mc` (voir Figure 4.7).

Figure 4.7

Aperçu du scénario de la scène principale.



Dans le calque des actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
//
import fl.ik.*;

//----- actions
// définition du squelette

IKManager.setStage(stage);
```



```

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Herbe");
squelette.registerElements(stage);

var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0)
➤ .getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).
➤ getChildAt(0).getChildAt(0).getChildAt(0);

// animation du squelette

var mouvement1:IKMover= new
IKMover(segment_colonneBas,segment_colonneBas.position);

stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
    var arrivee:Point=new Point(abeille_mc.x,abeille_mc.y);
    mouvement1.moveTo(arrivee);
}

var tweenAbeille:TweenMax;
tweenAbeille=TweenMax.to(abeille_mc,5,{x:900,delay:2,ease:Strong.easeInOut});
tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation);

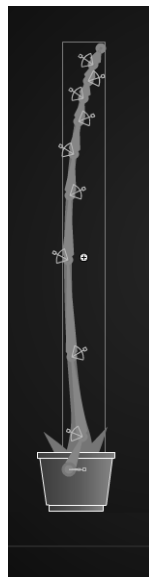
function suiteAnimation(evt:TweenEvent) {
    tweenAbeille=TweenMax.to(abeille_mc,8,{x:300,delay:0,ease:Back.easeInOut});
}

```

Comme dans l'exemple précédent, nous devons commencer par identifier l'ordre d'affichage des segments du squelette. Nous procédons, de même, en repérant le nom des occurrences depuis l'Inspecteur de propriétés et en déterminant le nombre de liaisons disponibles ainsi que la présence éventuelle de subdivisions multiples (voir Figure 4.8).

Figure 4.8

Aperçu de la structure du squelette nommé Squelette_Herbe.



Notre squelette ne comporte pas de subdivision. Il compte dix liaisons. Son nom d'occurrence est Squelette_Herbe. Les contraintes de rotation des liaisons ont été activées depuis l'Inspecteur de propriétés.

Ces contraintes ont plus particulièrement été marquées sur la base de l'ossature afin de renforcer l'effet d'enracinement et d'inertie pour la partie proche du pot, et donner un effet de pousse légère au sommet de la tige. Des contraintes identiques sur toute la longueur auraient donné une impression de structure entièrement molle ou totalement rigide, selon les valeurs d'angle enregistrées.

Dans les actions, nous importons d'abord les classes requises :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
//
import fl.ik.*;
```

En plus de la classe `ik` utilisée pour la cinématique inverse, nous importons la classe `TweenMax` (`gs`) pour animer le déplacement de l'abeille.

Plus loin, nous définissons le squelette avec `IKManager` et `IKArmature`, comme vu dans la section précédente.

```
//----- actions
// définition du squelette

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Herbe");
squelette.registerElements(stage);
```

Puis, comme nous l'avons vu précédemment, nous définissons le segment à animer:

```
var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0)
➤ .getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).
➤ getChildAt(0).getChildAt(0).getChildAt(0);
```

Ici, le segment que nous ciblons est le dernier de la chaîne. Elle en compte dix. Nous répétons donc dix fois le ciblage progressif jusqu'à atteindre notre cible.

Puis, nous animons l'objet en liant, cette fois-ci, ses coordonnées à celles de l'abeille, par l'intermédiaire d'une transition `TweenMax` :

```
// animation du squelette

var mouvement1:IKMover= new
IKMover(segment_colonneBas,segment_colonneBas.position);

stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
    var arrivee:Point=new Point(abeille_mc.x,abeille_mc.y);
    mouvement1.moveTo(arrivee);
}
var tweenAbeille:TweenMax;
tweenAbeille=TweenMax.to(abeille_mc,5,{x:900,delay:2,ease:Strong.easeInOut});
```

Lorsque l'abeille se déplacera à partir de sa position `x` actuelle (inférieure à 0) vers la position définie dans la transition, soit 900 px, la fonction associée à la classe `Event` et à l'événement `ENTER_FRAME` va permettre de repositionner le dernier segment selon la position `x`

de l'abeille en mouvement. Les segments étant reliés, et la méthode `MoveTo` répercutant les valeurs de positionnement vers les autres liaisons en fonction de leurs contraintes respectives, l'herbe haute va se courber pour suivre l'abeille dans son déplacement.

Nous terminons avec un enchaînement de transition :

```
tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation);

function suiteAnimation(evt:TweenEvent) {
    tweenAbeille=TweenMax.to(abeille_mc,8,{x:300,delay:0,ease:Back.easeInOut});
}
```

Ici, une seconde interpolation succède à la première, grâce à l'événement `COMPLETE` associé à la classe `TweenEvent`. Cette transition fait revenir, en marche arrière, l'abeille, de sorte qu'elle se place juste au-dessus de la plante. La fonction `activerMouvement1` qui demeure toujours active permet à la plante animée de continuer de suivre l'abeille jusqu'à son point arrêt. Tout nouveau mouvement que fera l'abeille par la suite provoquera un nouveau mouvement de l'herbe (voir Figure 4.9 à 4.12).

Figure 4.9

Aperçu
de l'animation
organique (1/4).



Figure 4.10

Aperçu
de l'animation
organique (2/4).



Figure 4.11

Aperçu
de l'animation
organique (3/4).

**Figure 4.12**

Aperçu
de l'animation
organique (4/4).

**À retenir**

- Il est possible de programmer l'animation de formes vectorielles avec la classe `ik`.
- Une liaison d'un squelette de forme vectorielle peut suivre un objet en mouvement.
- Il est possible de combiner des animations de type TweenMax avec des transitions de la classe `IKMover`.

Basculer du mode animation programmée au mode exécution

La programmation d'une animation de squelette requiert, pour le squelette actif de la scène courante, que le menu Type de la catégorie Options soit défini sur Exécution. Mais, une fois que l'animation est programmée, il est possible de rendre la main à l'utilisateur pour lui autoriser de déplacer lui-même les segments des objets animés.

Dans cette section, nous présentons comment basculer du mode d'animation programmée au mode 100 % interactif de l'affichage de l'armature, avec la classe IKManager (voir Figure 4.13).

Figure 4.13

L'utilisateur prend le contrôle du positionnement des liaisons.



Exemples > ch4_programmationDeSquelettes_3.fla

Le document que nous utilisons présente la structure suivante : dans la scène, nous retrouvons les mêmes éléments que dans la section précédente.

Dans le calque des actions, nous pouvons lire le code suivant, basé sur l'exemple précédent, mais nous y avons ajouté les commandes nécessaires au basculement de la propriété :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
//
import fl.ik.*;

//----- actions
// définition du squelette
```

```

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Herbe");
squelette.registerElements(stage);

var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0)
➤ .getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).
➤ getChildAt(0).getChildAt(0).getChildAt(0);

// animation du squelette

var mouvement1:IKMover= new
IKMover(segment_colonneBas,segment_colonneBas.position);

stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
    var arrivee:Point=new Point(abeille_mc.x,abeille_mc.y);
    mouvement1.moveTo(arrivee);
}

var tweenAbeille:TweenMax;
tweenAbeille=TweenMax.to(abeille_mc,5,{x:900,delay:2,ease:Strong.easeInOut});
tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation);

function suiteAnimation(evt:TweenEvent) {
    tweenAbeille=TweenMax.to(abeille_mc,8,{x:300,delay:0,ease:Back.easeInOut});
    tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation2);
}

// interrompre l'animation et basculer en mode interactif

var nombreDeBoucle:Number=1;
var dureeBoucle:Number=1000;
var boucle:Timer=new Timer(dureeBoucle,nombreDeBoucle);

function suiteAnimation2(evt:TweenEvent) {
    IKManager.trackAllArmatures(true);
    boucle.start();
    boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
}

function lancerBoucle(evt:TimerEvent) {
    stage.removeEventListener(Event.ENTER_FRAME, activerMouvement1);
}

```

Pour détecter la fin de l'animation, nous utilisons un nouvel écouteur attaché à l'objet animé, en l'occurrence, il s'agit de la seconde animation de l'abeille définie dans la fonction `suiteAnimation1`. Lorsque cette animation est achevée, nous programmons l'exécution d'une nouvelle fonction `suiteAnimation2`. Dans cette fonction, trois instructions sont ajoutées :

```

function suiteAnimation2(evt:TweenEvent) {
    IKManager.trackAllArmatures(true);
    boucle.start();
    boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
}

```

La première reprend la classe `IKManager` employée au début du code pour spécifier, à travers la propriété `trackAllArmatures`, que le déplacement de toute l'armature par l'utilisateur est désormais actif (`true`), selon les mêmes contraintes de rotation, définies initialement depuis l'Inspecteur de propriétés, que pour l'animation programmée.

Les deux suivantes activent d'abord un chronomètre de type `Timer` défini plus haut. Ces instructions associent, à ce même chronomètre, une nouvelle fonction qui interrompt l'animation de la liaison `activerMouvement1` dont la position est rattachée à celle de l'abeille, grâce à la méthode `removeEventListener` :

```
function lancerBoucle(evt:TimerEvent) {  
    stage.removeEventListener(Event.ENTER_FRAME, activerMouvement1);  
}
```

Notez que nous aurions pu interrompre la fonction directement dans la fonction `suiteAnimation2`, avec la même méthode `removeEventListener`. Mais, en procédant de la sorte, l'animation peut rendre un peu brutal l'arrêt du positionnement des éléments de liaison. En effet, l'animation est gérée par la classe `IKMover` et le moteur utilise un repositionnement qui comporte un algorithme d'accélération. L'animation aurait alors été interrompue au moment où le `TweenMax` s'achève, même si l'interpolation `IKMover` n'était pas terminée (du fait de l'amortissement). Afin que l'animation s'arrête de manière plus naturelle, nous avons recours à un `Timer`. Ceci laissera le temps à l'animation de l'abeille de terminer l'effet d'amortissement de son arrêt. Ensuite l'utilisateur pourra prendre la main.

À retenir

- Vous pouvez, à l'issue d'une animation programmée, rendre la main à l'utilisateur, pour lui permettre de déplacer lui-même les liaisons du squelette de l'objet animé.
- Les transitions `IKMover` utilisent un repositionnement qui comporte un algorithme d'accélération et obligent, pour les interrompre, d'utiliser un chronomètre de type `Timer` afin d'éviter que la rupture ne soit trop sèche.
- Pour basculer d'un mode d'exécution à un autre, nous utilisons la classe `IKManager`.

Activer un squelette chargé dans un SWF

Lorsqu'un document SWF qui comporte un squelette en mode Exécution est importé dans un nouveau document SWF, le squelette n'est pas actif dans l'interface de ce nouveau document. Cela s'explique par le fait que seul le document qui contient le squelette est compilé avec les objets d'affichages qui gèrent ce squelette. Même, en construisant un nouveau squelette dans le fichier appelant, le squelette importé ne réagit toujours pas. Il en résulte que tout document, dépourvu d'une action spécifique sur l'importation des squelettes, ne peut exécuter le squelette contenu dans tout document importé. Il est donc nécessaire de reconfigurer le document appelant, de sorte qu'il puisse exécuter correctement le fichier appelé.



Les noms du fichier appelant. Suite à un bogue du moteur Flash, il est possible que vous ne parveniez pas à importer un document si vous placez des caractères non ascii dans le nom du fichier. Utilisez de préférence des lettres collées, sans séparateur tiret -]. Pour séparer les mots, préférez le underscore ou tiret du bas (_). Les tirets peuvent ne pas être admis dans certaines configurations.

Dans cette section, nous importons un document SWF contenant un squelette de forme, publié en mode Exécution et le réactivons pour l'exécuter dans ce premier document.



Exemples > ch4_programmationDeSquelettes_4 fla
Exemples > ch4_programmationDeSquelettes_4a fla

Le document que nous utilisons se nomme "ch4ProgrammationDeSquelettes4 fla". Dans la scène, rien ne figure sinon le calque fond_mc (voir Figure 4.14).

Figure 4.14

Aperçu du scénario de la scène principale.



Dans le calque des actions, nous pouvons lire le code suivant :

```
//----- initialisation
import fl.ik.*;

//----- chargement
var chemin:URLRequest = new URLRequest("ch4-programmationDeSquelettes-4a.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);

chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);

var squelette:IKArmature;

function afficher(Evt:Event){
    addChild(chargeur);
    //
    IKManager.setStage(stage);
    squelette = IKManager.getArmatureByName("Squelette_Herbe");
    squelette.registerElements(stage);
    IKManager.trackAllArmatures(true);
}
```

D'abord, nous importons la classe ik pour reconstituer le squelette :

```
//----- initialisation
import fl.ik.*;
```


Puis, nous définissons un nouveau chargeur afin d'importer le fichier SWF contenant le squelette :

```
//----- chargement
var chemin:URLRequest = new URLRequest("ch4-programmationDeSquelettes-4a.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);
chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);
```

Lorsque le chargement du document est terminé, nous commençons par définir un espace mémoire pour la création d'une armature :

```
var squelette:IKArmature;
```

Puis, nous exécutons la fonction qui affiche le contenu et réactive l'armature du document importé :

```
function afficher(Evt:Event){
    addChild(chargeur);
    //
    IKManager.setStage(stage);
    squelette = IKManager.getArmatureByName("Squelette_Herbe");
    squelette.registerElements(stage);
    IKManager.trackAllArmatures(true);
}
```

À la suite de l'instruction `addChild`, la fonction place l'affectation du moteur de squelette sur la scène courante avec `setStage()`. L'instruction `getArmatureByName()` permet de pointer sur le squelette en utilisant le nom d'occurrence utilisé dans le SWF importé. Puis, elle mémorise les propriétés du squelette sur la scène globale avec `registerElements(stage)` et l'active avec `trackAllArmatures` passé sur `true`. C'est la dernière instruction qui permet de réactiver le mode d'affichage Exécution.

Cette fois, en publiant le document SWF, nous pouvons constater que le document importé est activé. Il est donc possible de déplacer les liaisons avec le pointeur et la plante préserve les contraintes définies dans le document importé (voir Figure 4.15).

Figure 4.15

Aperçu du SWF importé et exécuté.



À retenir

- Il est possible de réactiver un squelette désactivé automatiquement, après l'avoir importé dans un nouveau document SWF. Pour cela, vous devez réactiver le squelette par programmation dans le document appelant, afin de réimplanter le moteur d'exécution du squelette exclusif à l'API de Flash, lors de la publication du nouveau document.

Synthèse

Dans ce chapitre, vous avez appris à programmer des animations à partir de squelettes créés manuellement dans l'interface auteur de Flash. Vous avez appris à créer des animations mécaniques et organiques en exploitant des structures à base de symboles ou de formes. Vous avez également appris à programmer les animations de squelette en utilisant des objets et/ou le pointeur de la souris comme guide de mouvement. Vous savez enfin activer le mode d'exécution une fois l'animation achevée, et permettre à l'utilisateur de reprendre la main sur l'armature à la fin d'une interpolation. Vous êtes en mesure à présent de réaliser des animations de structure complexes et encore une fois dynamiques.

5

Les galeries d'images

Introduction

Pour gérer une galerie d'images, vous pouvez naturellement la travailler dans le scénario et jouer l'animation, mais il est préférable d'externaliser les contenus. D'abord, vous optimisez confortablement le poids de l'application. Ensuite, vous en simplifiez la maintenance. Dans ce contexte de galerie d'images externalisées, il suffit alors de remplacer les visuels importés par d'autres du même nom, pour automatiquement mettre à jour les images de l'animation, sans avoir à rééditer le Flash.

Pour construire une galerie d'images, nous devons d'abord apprendre à charger ces images progressivement, puis, à organiser leur affichage en les appelant directement un répertoire externe, et en y associant des informations textuelles, telles que des titres ou des légendes. Pour améliorer la gestion des contenus, nous verrons qu'il peut être intéressant de centraliser ces informations dans un fichier XML de sorte à simplifier la maintenance du projet une fois celui-ci publié.

Nous aborderons aussi la manière de rendre ces développements plus pertinents, d'autoriser l'interaction sur chaque objet généré dynamiquement et d'associer à chaque image, un lien, une fonctionnalité de zoom ou toute autre action.

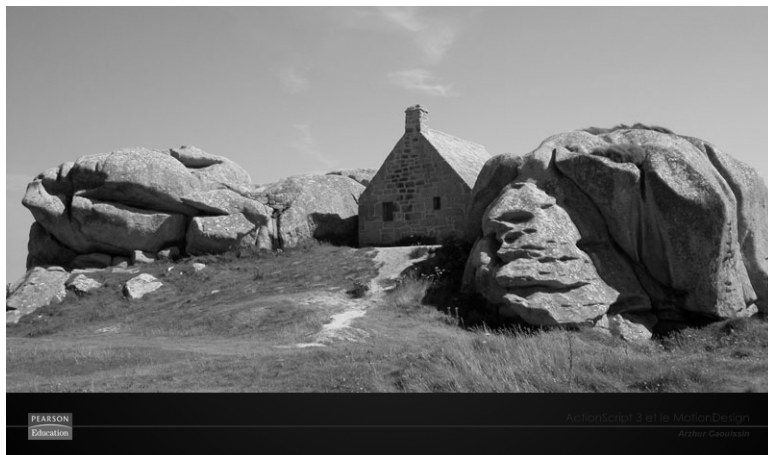
La détection de bogues éventuels pouvant survenir suite à une erreur de chargement ou d'exécution du programme, nous traiterons la manière de placer des contrôles afin d'éradiquer les risques de plantage du programme. Enfin, nous évoquerons l'intégration dynamique des données que vous pourriez extraire d'une base de type MySQL ou de tout autre système d'informations, grâce à vos connaissances acquises sur la gestion d'un flux XML.

Afficher des images externalisées et aléatoires

Dans cette première section, nous abordons le chargement d'images externalisées. Cela nous permet d'alléger grandement le poids des documents SWF que nous créons. Nous abordons aussi la gestion de l'affichage d'une image avec le paramètre `random` de sorte que l'animation publiée affiche un visuel ou un autre et ce, de manière entièrement aléatoire (voir Figure 5.1).

Figure 5.1

Un des aperçus
du document à la
publication.



Exemples > ch5_galleriesImages_1.flu

Dans la scène du document "ch5_galerieImages_1.flu", au-dessus du calque fond_mc, apparaît un calque masqué nommé cible_mc (voir Figure 5.2). Ce calque affiche un symbole de type MovieClip du même nom. Ce symbole est vide et sert de conteneur pour importer les images appelées dynamiquement par ActionScript.

Figure 5.2

Aperçu du
scénario de la
scène principale.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
// CHARGEMENT
var chargeurFond:Loader = new Loader();
var valeurAleatoire:Number=Math.round(Math.random()*2);
var urlFond:URLRequest=new URLRequest("images/coteBretonne/photo"+valeur
➔ Aleatoire+".jpg");
chargeurFond.load(urlFond);

// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurFond);
}
```

D'abord, une série de variables initialise le chargement :

```
var chargeurFond:Loader = new Loader();
```

La variable chargeurFond réserve l'emplacement mémoire pour la définition d'un chargeur (:Loader). Le signe égal (=) affecte une valeur qui désigne, ici, l'instanciation d'un nouvel objet Loader (= new Loader()).

Plus loin, une variable de type nombre (`valeurAleatoire`) définit une valeur aléatoire comprise entre 0 et 2 inclus. Celle-ci est obtenue grâce à la classe `Math.random()`. La classe `Math.round()` utilisée également permet d'arrondir la valeur obtenue à un chiffre entier. Pour combiner les deux méthodes, nous imbriquons la première dans la seconde :

```
var valeurAleatoire:Number=Math.round(Math.random()*2);
```

C'est cette valeur que nous récupérerons plus tard et qui déterminera l'image à charger.

Générer une valeur aléatoire

Pour générer une valeur aléatoire, nous utilisons la classe `Math.random()` multipliée par une valeur. Cette valeur correspond au seuil limite (supérieur) du nombre que l'on souhaite obtenir. Par exemple, `Math.random()*5` renvoie une valeur comprise entre 0 et 5.

Mais attention, la valeur obtenue, même multipliée, reste décimale, car en réalité, `Math.random()` génère une valeur comprise entre 0,00... et 0,9999... En la multipliant, nous agrandissons donc uniquement les valeurs décimales en d'autres valeurs décimales, mais à une échelle plus grande. Par exemple, si l'on considère :

```
Math.random()*5;
```

et que la méthode `Math.random()` renvoie 0.7. Alors, nous obtenons :

$$0,7 \times 5 = 3,5$$

Arrondir une valeur

Pour obtenir un chiffre entier, nous utilisons l'une des méthodes suivantes :

- Pour arrondir à l'entier le plus proche, on utilise `Math.round()` : `Math.round(Math.random()*5)`.
- Pour arrondir à l'entier supérieur le plus proche, on utilise `Math.ceil()` : `Math.ceil(Math.random()*5)`.
- Pour arrondir à l'entier inférieur le plus proche, on utilise `Math.floor()` : `Math.floor(Math.random()*5)`.

Ainsi, la méthode `Math.round`, en dessous de 0.5, arrondit à 0, au dessus de 0.5, arrondit à 1. La méthode `Math.ceil`, de 0.01 à 0.9, arrondit à 1, et la méthode `Math.floor`, de 0.01 à 0.999, arrondit à 0... Testez par exemple ceci pour identifier clairement le résultat obtenu par chacune des trois méthodes :

```
trace("floor")
for (var i:Number=0;i<20;i++){
    trace(Math.floor(Math.random()*2));
}
trace("round")
for (var i:Number=0;i<20;i++){
    trace(Math.round(Math.random()*2));
}
trace("ceil")
for (var i:Number=0;i<20;i++){
    trace(Math.ceil(Math.random()*2));
}
```

Une fois la valeur aléatoire définie, une autre variable, `urlFond`, stocke le chemin d'accès à l'image. L'image est localisée dans un dossier nommé "coteBretonne" situé dans le répertoire "images" qui accompagne les documents Flash des exemples du livre :

```
var urlFond:URLRequest=new URLRequest("images/coteBretonne/
photo"+valeurAleatoire+".jpg");
```

Le nom de la photo appelée est construit de manière dynamique. Nous concaténons à la volée notre valeur aléatoire entre la racine du nom et son extension (racine + valeurAleatoire + extension). Selon la valeur obtenue par la variable, le chemin enregistré pour le chargement de l'image sera "images/coteBretonne/photo0.jpg", "images/coteBretonne/photo1.jpg" ou "images/coteBretonne/photo2.jpg".



Définir les chemins pour les requêtes externes. Lorsque vous appelez un contenu situé à l'extérieur d'un document Flash (une image, un autre fichier SWF, un flux XML, un son, une vidéo, etc., sauf pour les classes `.as` qui sont compilées à la publication), il est important de déterminer l'emplacement du contenu appelé par rapport à l'emplacement de la page HTML qui exécute le fichier Flash, et non par rapport au fichier Flash lui-même. En effet, si le fichier Flash est enregistré dans un emplacement différent de la page qui le contient, le chemin se référant à ce document Flash risque d'être invalide. Considérez toujours que la page HTML qui exécute le Flash modifie de ce fait la position relative de l'animation Flash, en la définissant par rapport à la page HTML elle-même.

Une fois le chemin défini pour le chargement de l'image, nous indiquons au chargeur (chargeurFond), à l'aide de la méthode `load`, de charger cette image (`urlFond`).

```
chargeurFond.load(urlFond);
```

Plus loin, un écouteur est attaché au chargeur `chargeurFond` et appelle une fonction :

```
// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurFond);
}
```

L'écouteur est attaché à l'objet `LoaderInfo` du chargeur et non directement au chargeur lui-même. La propriété `contentLoaderInfo` appartient en effet à l'objet `Loader` et permet d'y lire des informations relatives à la progression du chargement. Pour ajouter des instructions d'affichage suite au chargement, nous devons donc introduire cet objet dans le gestionnaire d'événements, ce qui donne :

```
chargeurFond.contentLoaderInfo.addEventListener
(Event.COMPLETE,chargementCOMPLET);
```

Lorsque le chargement est complet (`Event.COMPLETE`), c'est-à-dire lorsque l'image appelée avec la méthode `load` est entièrement chargée sur le poste client, l'écouteur exécute une fonction que nous avons ici nommée `chargementCOMPLET` :

```
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurFond);
}
```

Cette fonction affiche l'objet chargé directement dans le MovieClip nommé `cible_mc` qui sert de conteneur (ou l'ajoute comme enfant à la liste d'affichage), à l'aide de la méthode `addChild()` :

```
Cible_mc.addChild(chargeurFond);
```

En affichant l'élément chargé dans un symbole de type MovieClip, nous lui faisons bénéficier des propriétés afférentes aux MovieClip. L'objet importé peut donc, par ce conteneur, être animé et contrôlé par ActionScript (position, effets de couleur, affichage, filtres, animations, etc.).



Doit-on toujours placer un écouteur sur un chargeur pour activer l'affichage d'un contenu chargé dynamiquement ? Il est possible, lorsque vous réalisez le document localement, de vous passer de l'écouteur attaché au chargeur et d'invoquer directement l'affichage (`addChild`) suite au chargement avec la méthode `load`. Le fichier placé localement est déjà présent sur votre poste et ne requiert pas, dans ce cas, d'être préalablement téléchargé avant d'être affiché. Il est affiché directement, car il est déjà disponible. Mais, pour tout contenu destiné à une publication en ligne, pour un site donc, il convient de passer par l'écouteur qui assurera l'affichage seulement une fois le contenu entièrement chargé. Le cas échéant, les contenus appelés dynamiquement seraient considérés par le lecteur Flash comme inexistant au moment de l'exécution des scripts attachés à ces objets, ce qui pourrait nuire à la bonne exécution de votre programme. Pour placer une action alternative en cas de problème lié au chargement, consultez aussi la section consacrée à la gestion des erreurs au chargement, en fin de chapitre.

À retenir

- Pour optimiser le poids d'un document qui affiche des images de grande taille, il est souhaitable d'externaliser ces images à l'aide d'un chargeur.
- Les paramètres du chargeur permettent de définir de manière dynamique les contenus à charger. Nous pouvons de ce fait avoir recours à une variable aléatoire pour construire le chemin de chargement.
- Pour appliquer une valeur aléatoire, nous utilisons la classe `Math.random` et nous arrondissons à un entier la valeur décimale obtenue avec les classes `Math.round`, `Math.ceil` ou `Math.floor`.
- Les chemins des éléments chargés dynamiquement sont relatifs aux pages qui contiennent l'animation Flash.
- Il convient d'utiliser un écouteur pour attendre la fin du chargement d'un contenu avant de l'afficher, à défaut de quoi, l'application développée pourrait ne pas fonctionner normalement.

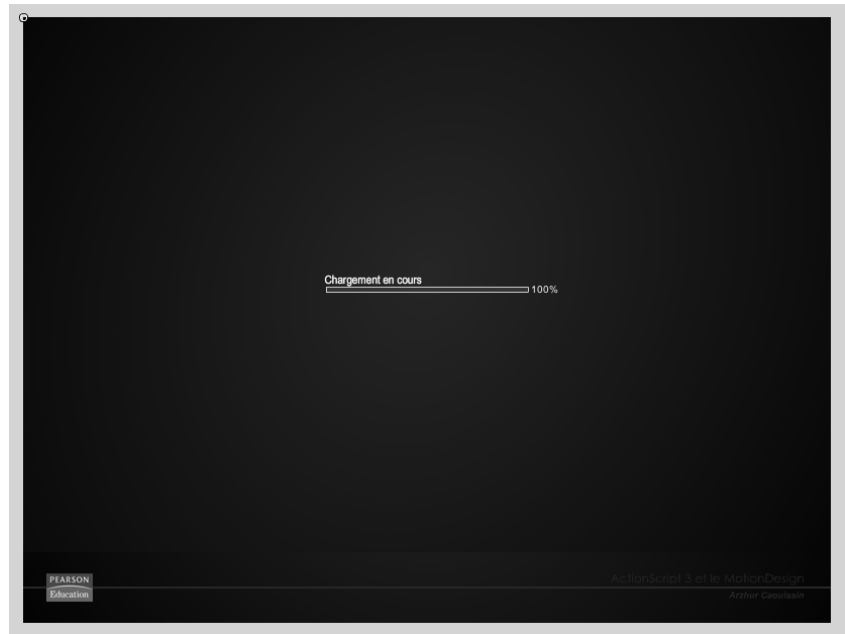
Réaliser une jauge de chargement

Lorsque le contenu chargé dynamiquement est lourd (plus de 100 Ko), il est souhaitable de signaler la progression du chargement. Pour ce faire, nous utilisons une jauge signalétique afin d'avertir l'utilisateur que l'absence d'affichage ne désigne pas un bogue ou une erreur de sa part, mais qu'il faut attendre la fin du chargement pour en visualiser le contenu.

Dans cette section, nous ajoutons donc une jauge de progression au chargement mis en œuvre à la section précédente (voir Figure 5.3).

Figure 5.3

Aperçu de la jauge de chargement.



Exemples > ch5_galleriesImages_2.fla

Dans la scène du document "ch5_galleriesImages_2.fla", au-dessus du calque `fond_mc`, apparaît un calque masqué nommé `cible_mc`. Ce calque affiche le même symbole `cible_mc` vide, observé précédemment. Au-dessus du masque figure un autre calque, `chargement_mc`, qui affiche une jauge de progression (voir Figure 5.4). À l'intérieur de ce nouveau symbole, nous distinguons, clairement répartis vers les calques (voir Figure 5.5), un autre symbole, `barre_mc`, qui contient une forme graphique rectangulaire et dont le centre géométrique est positionné à gauche (pour que l'objet se déforme vers la droite, dans le sens de la lecture de la progression) (voir Figure 5.6), ainsi qu'un texte dynamique nommé `pourcentage_txt`. Les autres éléments sont purement figuratifs.

Figure 5.4

Aperçu du scénario de la scène principale.



Figure 5.5

Aperçu du scénario
du symbole
chargement_mc.

**Figure 5.6**

Aperçu du symbole
barre_mc dont le
centre géométrique
est situé à gauche.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
// CHARGEMENT
var chargeurFond:Loader = new Loader();
var valeurAleatoire:Number=Math.round(Math.random()*2);
var urlFond:URLRequest=new URLRequest("images/coteBretonne/photo"+valeur
➤ Aleatoire+".jpg");
chargeurFond.load(urlFond);

// PROGRESSION
chargeurFond.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➤ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current
➤ Target.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
}

// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurFond);
    chargement_mc.visible=false;
}
```

Comme à la section précédente, nous retrouvons d'abord les variables qui activent le chargement. Ensuite, deux écouteurs sont attachés à la même propriété contentLoaderInfo.

Le premier écouteur exécute une fonction durant la progression du chargement (PROGRESS), alors que le second en exécute une autre une fois ce chargement terminé (COMPLETE). Revenons sur le premier écouteur :

```
chargeurFond.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➤ chargementENCOURS);
```

Dans le gestionnaire d'événements, nous pouvons lire ProgressEvent et non simplement Event. Seule la classe ProgressEvent permet de disposer de l'événement PROGRESS qui désigne la période durant laquelle le chargement s'effectue et autorise, seul, l'exécution d'une fonction pendant ce chargement.

La fonction appelée pendant le chargement est `chargementENCOURS` :

```
function chargementENCOURS(evt:ProgressEvent) {
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.currentTarget.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
}
```

Dans cette fonction, nous commençons par définir une variable de type nombre qui enregistre une valeur correspondant à la progression du chargement. Cette valeur décimale est comprise entre 0 (0 % de progression) et 1 (100 % de progression). Elle est déterminée en calculant, pour l'objet en cours de chargement (`evt.currentTarget`), le nombre de bytes déjà chargés divisé par le nombre de bytes total à charger.

Par exemple, si le fichier pèse 60 Ko (61 440 bytes) et que 15 Ko sont actuellement chargés (15 360 bytes), le pourcentage du chargement effectué est de 15 360/61 440, soit 0,25 (ou 25 %). Lorsque le chargement sera terminé, la valeur sera de 61 440/61 440, soit 1.

Suite à cela, la variable `valeurPourcentage` est utilisée pour définir l'échelle de déformation en X sur l'objet `barre_mc` qui représente la jauge de progression du symbole `chargement_mc` :

```
chargement_mc.barre_mc.scaleX=valeurPourcentage;
```

Plus loin, dans la même fonction, une autre instruction utilise cette valeur pour définir le texte à afficher dans le champ dynamique nommé `pourcentage_txt`, situé à l'intérieur du symbole `chargement_mc` :

```
chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
```

Le texte résulte de la variable `valeurPourcentage` augmentée et arrondie. Nous concaténons, avec le signe plus (+), le caractère pourcentage (%) pour que le chiffre obtenu soit accompagné à l'écran de cette unité de valeur.

Lorsque le chargement atteint la valeur de 100, le texte affiche donc 100 %, le symbole `barre_mc` retrouve son échelle initiale et la fonction `chargementCOMPLET` est alors exécutée :

```
// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurFond);
    chargement_mc.visible=false;
}
```

Dans cette fonction, en plus d'afficher le contenu chargé avec `addChild`, nous rendons invisible le symbole `chargement_mc`.

En publiant le document localement sur votre ordinateur, vous ne visualisez probablement pas la progression du chargement, car le fichier appelé est déjà présent sur votre poste et ne requiert donc aucune attente pour être affiché. Pour mieux vous rendre compte de l'efficacité de la jauge de chargement, dans le menu Affichage situé en haut de votre écran et disponible lors de la publication du document (Cmd+Entrée pour Mac ou Ctrl+Entrée pour Windows), activez l'option Simuler le téléchargement. Si le réglage de la bande passante est

correctement configuré, vous pourrez alors visualiser la progression du chargement comme si le fichier était en ligne.

À retenir

- Pour créer une jauge de chargement, nous utilisons `contentLoaderInfo`, une propriété spécifique associée à la classe `ProgressEvent`.
- La valeur obtenue pour déterminer la progression du chargement peut être distribuée à travers différentes propriétés pour redimensionner un symbole (jauge) ou modifier la valeur d'un texte (pourcentage), par exemple.
- Lorsque le chargement est terminé, nous affichons le contenu chargé et nous masquons la jauge de chargement.
- Pour animer l'échelle du symbole qui sert de jauge, il faut bien définir l'emplacement de son centre géométrique de manière à l'étirer correctement.

Réaliser une galerie d'images externalisées

Pour réaliser une galerie d'images à l'aide d'ActionScript, nous utilisons un chargeur. À chaque itération appelée en activant un bouton suivant ou retour, nous remplaçons l'image chargée par une nouvelle. Pour que la gestion de ce dispositif reste simple, nous organisons les fichiers et leurs noms de sorte que le processus puisse être facilement automatisé. Ainsi, si nous rassemblons les images de la galerie dans un même répertoire, et que nous nommons ces images avec des numéros correspondant respectivement à leur ordre d'apparition, et en commençant par zéro, il nous suffit alors d'appeler chaque image en incrémentant ou en diminuant une valeur passée en paramètre de l'URL d'un chargeur de contenu. Nous pouvons aussi utiliser cette valeur pour renseigner l'internaute sur le numéro d'image en cours d'affichage, par exemple. C'est ce que nous présentons dans cette section (voir Figure 5.7).

Figure 5.7

Aperçu de la galerie à la publication.





Exemples > ch5_galleriesImages_3 fla

Dans la scène principale du document "ch5_galleriesImages_3 fla", au-dessus du calque fond_mc, apparaît le symbole cible_mc. Il est vide et sert de conteneur pour les images de la galerie (voir Figure 5.8). Ce symbole possède déjà des propriétés, telles que l'application d'un filtre d'ombre portée. Nous retrouvons la jauge de chargement présentée à la section précédente, puis deux boutons (retour_btn et suivant_btn) qui servent à appeler les nouvelles images. Au-dessus, deux champs de texte dynamiques permettent d'inscrire pour chaque photo, un titre et une légende en fonction des itérations. Enfin, deux filets soulignent la composition, mais ne sont pas impliqués dans les actions.

Figure 5.8

Aperçu
du scénario de la
scène principale.



À l'intérieur de notre dossier de travail, le répertoire "images" contient un sous-répertoire nommé "galerie". Celui-ci contient 7 images respectivement nommées photo0.jpg, photo1.jpg, photo2.jpg, etc., de largeurs différentes (voir Figure 5.9).

Figure 5.9

Aperçu
des images
externalisées
de la galerie.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
//----- Chargement initial

// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
chargeurPhoto.load(cheminPhoto);

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➤ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
    chargement_mc.visible=true;
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded /
➤ evt.currentTarget.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
```

```

    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET1);
function chargementCOMPLET1(evt:Event) {
    cible_mc.addChild(chargeurPhoto);
    chargement_mc.visible=false;
    cible_mc.x=(stage.stageWidth/2)-(cible_mc.width/2);
    cible_mc.y=(stage.stageHeight/2)-(cible_mc.height/2)-25;
}

//----- Boutons de navigation

var i:Number=0;
var nombreDePhotos:Number=7;
titre_txt.text="Galerie Photo";
index_txt.text="Les marais salants de Guérande";

// Bouton SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK,afficherPhotoSuivante);

function afficherPhotoSuivante(Event:MouseEvent){
    i++;
    if (i>=nombreDePhotos) {
        i=0;
    }
    chargerPhoto();
}

// Bouton RETOUR
retour_btn.addEventListener(MouseEvent.CLICK,afficherPhotoRetour);

function afficherPhotoRetour(Event:MouseEvent){
    i--;
    if (i<0) {
        i=nombreDePhotos-1;
    }
    chargerPhoto();
}

function chargerPhoto () {
    var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo"+i+".jpg");
    chargeurPhoto.load(cheminPhoto);
    index_txt.text="Photo N° "+i;
}

```

La première partie de ce code reprend essentiellement le chargement initial d'une image, que nous avons abordé précédemment. L'URL cible simplement un nouveau dossier et affiche la première image de la série, nommée "photo0.jpg" :

```
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
```

Dans le gestionnaire PROGRESS, nous réactivons l'affichage de la jauge de sorte qu'elle puisse réapparaître aussi pour les images suivantes :

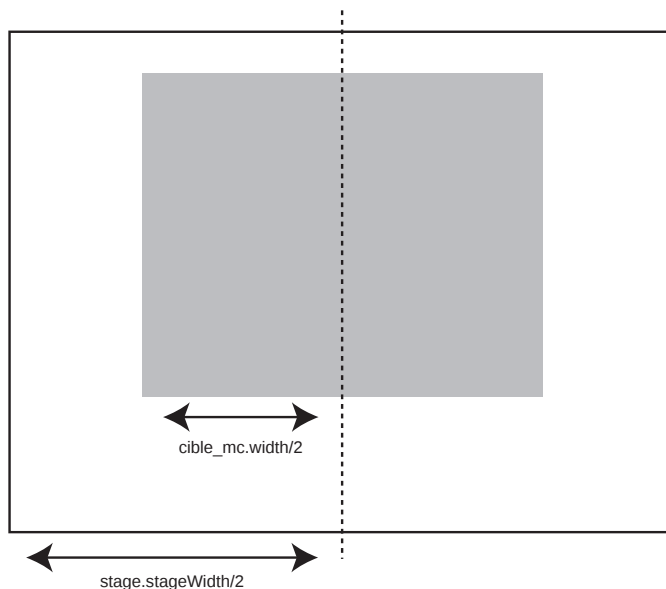
```
chargement_mc.visible=true;
```

Enfin, une fois le chargement complet, nous redéfinissons le positionnement du symbole conteneur `cible_mc` en X et Y. Dans notre contexte, cela permet de recentrer l'image dans la page quelle que soit sa largeur. Pour ce faire, nous spécifions, dans le premier groupe de parenthèses, que la position du conteneur correspond à la largeur de la fenêtre du document Flash, divisée par 2. Puis, nous retranchons, dans le deuxième groupe de parenthèses, la largeur du conteneur lui-même, divisée aussi par 2. Si nous ne divisons pas par 2, la cible serait collée à droite de la limite du document, car la valeur alors prise en compte pour caler l'objet serait la largeur de la scène moins celle de l'objet. En divisant par 2 chacune d'entre elles, puisque la moitié d'une largeur équivaut à son centre, nous centrons ce conteneur dans le document (voir Figure 5.10) :

```
cible_mc.x=(stage.stageWidth/2)-(cible_mc.width/2);
```

Figure 5.10

Schéma du positionnement du conteneur.



Notez que ce calcul ne peut se faire qu'à partir du moment où les dimensions de `cible_mc` sont connues, c'est-à-dire, uniquement lorsqu'une photo y est chargée. À défaut, c'est sa dimension initiale qui sera lue, soit 0 pixel de large. La position d'un conteneur est déterminée par rapport à son centre géométrique, toujours calé en haut et à gauche dans le cas d'un chargement dynamique. Elle serait effectivement décalée de la moitié de la largeur de l'image importée, si nous l'appliquions avant le chargement de l'image.

Le principe est décliné pour le positionnement vertical, en Y, avec toutefois un décalage de -25 pixels, pour rehausser le conteneur vers le sommet du document :

```
cible_mc.y=(stage.stageHeight/2)-(cible_mc.height/2)-25;
```



Centrer les images avec le composant UILoader. Il est possible de centrer les images chargées dynamiquement sans avoir à calculer leur position. Remplacez simplement le conteneur MovieClip `cible_mc` que nous utilisons dans cet exemple par un composant UILoader. Ce composant est disponible depuis la fenêtre des composants (Fenêtre > Composants), dans le groupe de composants

nommé "User Interface". Pour plus d'informations sur ce composant, reportez-vous à l'aide de Flash (F1) et saisissez dans le moteur de recherche de l'aide, le terme `UILoader` ou bien consultez directement l'URL de la notice de ce composant à cette adresse : http://help.adobe.com/fr_FR/ActionScript/3.0_UsingComponentsAS3/WS5b3ccc516d4fbf351e63e3d118a9c65b32-7f9d.html.

Plus loin dans le code, au niveau du commentaire Boutons de navigation, nous initialisons quelques valeurs :

```
//----- Boutons de navigation
var i:Number=0;
var nombreDePhotos:Number=7;
titre_txt.text="Galerie Photo";
index_txt.text="Les marais salants de Guérande";
```

En premier lieu, `i` désigne un nombre de valeur 0. À chaque clic sur le bouton suivant ou retour, nous incrémentons ou diminuons cette valeur de sorte qu'elle désigne, dans la liste des images, l'image correspondant à la valeur appelée. Pour démarrer, cette valeur est initialisée à 0 et désigne l'image "photo0.jpg". À chaque clic, cette valeur est donc modifiée. Lorsque la valeur atteint le seuil correspondant à la dernière image de la série, nous l'initialisons à 0, pour redémarrer la boucle. Inversement, lorsque le bouton retour appelle l'image qui précède la première image de la série, la boucle renvoie la valeur qui correspond à la dernière image (`i=nombreDePhotos-1`). Nous retranchons 1 à la valeur `nombreDePhotos` : la boucle démarrant à 0, si nous conservions la valeur initiale, nous obtiendrions une itération de trop en regard du nombre d'images disponibles, ce qui provoquerait une erreur d'affichage.

La deuxième variable `nombreDePhotos` indique le nombre de photos disponibles dans la galerie. Vous devez renseigner cette valeur manuellement. Nous verrons, en abordant le XML, que cette valeur peut être renseignée automatiquement. Actuellement, nous comptons 7 photos.

Enfin, nous initialisons les textes des champs dynamiques `titre_txt` et `index_txt` distribués sur la scène. Pour chaque bouton, nous ajoutons à présent un écouteur et une fonction :

```
// Bouton SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK,afficherPhotoSuivante);
```

Le bouton `suivant_btn` appelle la fonction `afficherPhotoSuivante` :

```
function afficherPhotoSuivante(Event:MouseEvent){
    i++;
    if (i>nombreDePhotos) {
        i=0;
    }
    chargerPhoto();
}
```

La fonction commence par incrémenter la valeur `i` (`i++`) préalablement initialisée à 0. La valeur `i` devient donc 1, au premier clic, puis 2 au second clic et ainsi de suite.

Ensuite, nous indiquons que si la valeur de `i` dépasse le nombre de photos disponibles (`i>nombreDePhotos`), nous la réinitialisons à 0. Ainsi, lorsque la boucle est terminée, c'est la première image qui est à nouveau affichée et la boucle peut continuer d'être incrémentée jusqu'à ce que la condition l'initialise de nouveau, et ainsi de suite.

À la fin du bloc d'instruction du premier bouton, nous faisons référence à une fonction développée plus loin (`chargerPhoto`). Nous y revenons.

Nous identifions ensuite une nouvelle fonction (`afficherPhotoRetour`) déclinaison de cette même fonction pour le bouton de retour :

```
// Bouton RETOUR
retour_btn.addEventListener(MouseEvent.CLICK,afficherPhotoRetour);
function afficherPhotoRetour(Event:MouseEvent){
    i--;
    if (i<0) {
        i=nombreDePhotos-1;
    }
    chargerPhoto();
}
```

Dans la fonction `afficherPhotoRetour`, nous inversons le sens de l'incréméntation en remplaçant les signes plus par moins (`i--`). La valeur ainsi diminue de 1 à chaque itération. Puis, si cette valeur atteint le seuil limite qui correspond à la première image (0), alors, nous l'initialisons à la valeur de la variable `nombreDePhotos` de sorte que l'animation boucle à nouveau sur elle-même.

À la fin du programme, nous plaçons une fonction autonome qui rassemble les instructions communes appelées par les deux boutons. Cela permet de simplifier le codage du projet en évitant les actions redondantes. De plus, cela facilitera la maintenance :

```
function chargerPhoto () {
    var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo"+i+".jpg");
    chargeurPhoto.load(cheminPhoto);
    index_txt.text="Photo N° "+i;
}
```

La fonction `chargerPhoto()` lance le chargement de l'image et modifie la valeur du champ de texte dynamique en fonction de la valeur de l'image chargée, définie par `i`.

Plus précisément, nous redéfinissons l'URL `cheminPhoto` appelée par le chargeur du départ, nommé `chargeurPhoto`. Nous utilisons la valeur de `i` pour déterminer le numéro de l'image à afficher, en intégrant cette valeur au sein de la chaîne de caractères de l'URL. Il est nécessaire, pour le bon fonctionnement du programme, que les images stockées dans le répertoire possèdent la même racine de nom (`photo`), et que la première d'entre elle se termine par le numéro 0 qui correspond à la première valeur de `i`.

L'ordre numérique en ActionScript

En ActionScript, la valeur du premier niveau de toute chaîne démarre toujours à 0. Un objet affiché sur la scène sera accessible avec `getChildAt(0)`. Le premier nœud d'un fichier XML sera appelé avec `documentXML.childNodes[0]`. Le premier élément d'un tableau sera accessible avec `nomDuTableau[0]`, et ainsi de suite. Pour homogénéiser les actions et simplifier la gestion d'actions répétitives, nous gérons aussi les images en associant au nom, de la première d'entre elles, la valeur 0. Il est alors plus simple de gérer les images en associant leur nom à une variable d'incréméntation, comme `i`, qui peut en même temps affecter d'autres méthodes ou conteneurs.

Nous rappelons alors le chargeur (`chargementphoto.load(cheminPhoto)`) afin qu'il relance le chargement à partir de l'URL nouvellement désignée.

Il n'est pas nécessaire de rappeler la méthode `addChild` car l'objet `chargeurPhoto` est déjà présent dans la liste d'affichage (donc sur la scène), grâce au premier `addChild` invoqué lors du chargement de la première image. De même, il est inutile de relancer les écouteurs pour les événements `PROGRESS` et `COMPLETE` qui demeurent toujours actifs tant que nous ne les avons pas neutralisés à l'aide de la méthode `removeEventListener`.

En publiant la galerie sur un serveur distant, vous remarquerez que la jauge réapparaît pour chaque nouveau chargement parce que nous avons réactivé la propriété `visible` sur `true` à l'exécution du chargement. De même, la jauge disparaît lorsque vous affichez une deuxième fois chaque image, car celles-ci sont maintenant chargées dans le cache de votre navigateur. Leur chargement devient donc instantané.

Enfin, toujours dans cette fonction, nous modifions le contenu du champ de texte dynamique `index_txt` en y inscrivant le numéro de la photo chargée, selon le même principe que pour la définition de l'URL.

À retenir

- Il est possible de réaliser une galerie dynamique d'images en jouant sur la manière de nommer les fichiers appelés. Par exemple, en mixant une racine commune avec un nombre qui reflète l'ordre d'apparition.
- Pour appeler différents contenus externalisés ou modifier des valeurs, nous pouvons utiliser une variable de type nombre qu'il suffit d'incrémenter à chaque itération pour générer de nouvelles valeurs.
- Un gestionnaire d'événements est toujours actif tant que celui-ci n'a pas été neutralisé par la méthode `removeEventListener`.
- Il est possible de centrer une image chargée dynamiquement en calculant sa position par rapport aux dimensions de la fenêtre du document Flash (`stage`) ou en utilisant le composant `UILLoader`.

Réaliser une galerie d'images avec XML

Le recours à un fichier XML, pour le développement d'une galerie photo comme nous le proposons ici, permet de centraliser les informations rattachées à chaque visuel dans un seul document, éditable au format texte, et de ne plus avoir à publier à nouveau le document Flash lorsqu'une donnée est modifiée.

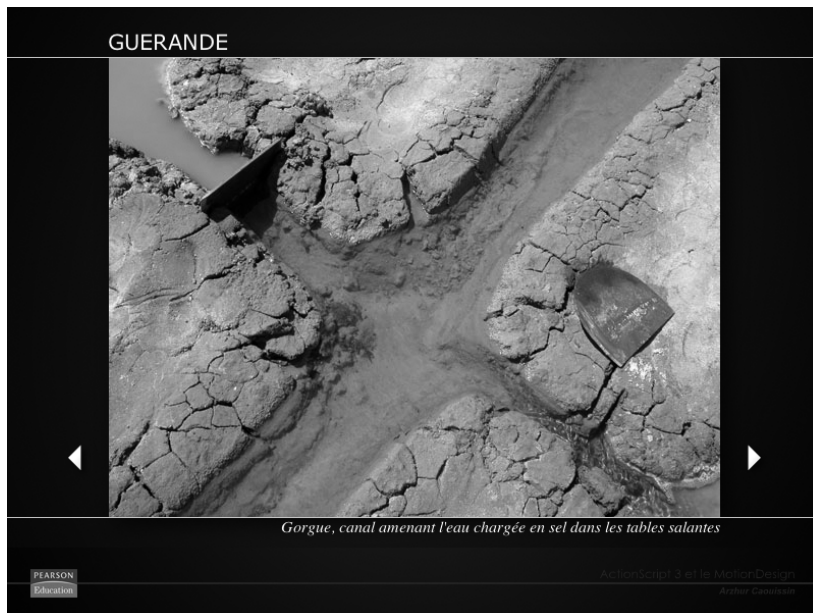
La gestion d'une animation Flash avec un flux XML vous permet aussi de travailler plus facilement en équipe avec un développeur back-office par exemple (qui gère des systèmes de gestion de contenu côté serveur, avec les langages PHP/MySQL par exemple, et peut rendre facilement disponibles ses requêtes sous la forme de flux XML).

La structure d'un document XML autorise enfin d'associer, pour chacun des éléments, un nombre indéterminé de données, sans avoir à les traiter toutes systématiquement.

Dans cette section, nous présentons une galerie photo associée à un fichier XML contenant les références aux images, les titres ainsi qu'une légende pour chacune d'entre elles. En cliquant sur les boutons suivant et retour, nous évoluons directement au cœur de l'arborescence du fichier XML alors importé (voir Figure 5.11).

Figure 5.11

Aperçu de la galerie à la publication.

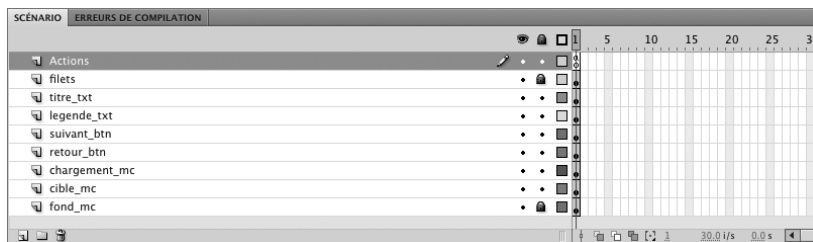


Exemples > ch5_galleriesImages_4.flu

Dans la scène principale du document "ch5_galleriesImages_4.flu", au-dessus du calque fond_mc, apparaît le même symbole cible_mc vide, qui sert de conteneur pour les images de la galerie. Les autres éléments sont similaires. Deux champs de texte dynamiques titre_txt et legende_txt sont distribués de part et d'autre de la zone d'affichage cible_mc. Deux boutons suivant_btn et retour_btn permettent de contrôler la navigation. Enfin, le chargeur développé en début de chapitre demeure présent et reste également actif pour chaque nouvelle entrée appelée (voir Figure 5.12).

Figure 5.12

Aperçu du scénario de la scène principale.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
//----- Chargement initial PHOTO AGRANDIE
// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
chargeurPhoto.load(cheminPhoto);

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➡ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
    chargement_mc.visible=true;
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current
➡ Target.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET1);
function chargementCOMPLET1(evt:Event) {
    cible_mc.addChild(chargeurPhoto);
    chargement_mc.visible=false;
    cible_mc.x=stage.stageWidth/2-cible_mc.width/2;
    cible_mc.y=stage.stageHeight/2-cible_mc.height/2-25;
}

//----- Chargement du XML

var i:Number=0;

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLComplet);

function XMLComplet(evt:Event){
    var donneesXML:XML=new XML(evt.target.data);
    var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();

    // RETOUR
    retour_btn.addEventListener(MouseEvent.MOUSE_DOWN, retourDOWN);
    function retourDOWN(evt:MouseEvent) {
        i--;
        if (i<0) {
            i=NombreDeNoeudsDansLeXML-1;
        }
        distribuerValeurs();
    }
}

// SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
function suivant(evt:MouseEvent) {
```

```

        i++;
        if (i==NombreDeNoeudsDansLeXML) {
            i=0;
        }
        distribuerValeurs();
    }
    function distribuerValeurs () {
        titre_txt.htmlText=donneesXML.photo[i].titre.toString();// titre
        legende_txt.htmlText=donneesXML.photo[i].legende.toString();// légende
        cheminPhoto=new URLRequest(donneesXML.photo[i].image.toString());
        ➡ // image agrandie
        chargeurPhoto.load(cheminPhoto);
    }
}

```

Pour comprendre le principe de la gestion d'un flux XML, nous devons retenir l'idée que les données centralisées dans le fichier XML ne peuvent être distribuées dans le Flash qu'une fois le XML entièrement chargé. C'est pourquoi nous mettons les contenus en forme suite à l'événement COMPLETE.

Pour traiter les données issues d'un fichier XML, nous disposons alors de deux méthodes. Soit nous utilisons directement les valeurs chargées par le XML au sein de la fonction liée au chargement du XML. Soit nous créons des variables globales, c'est-à-dire que nous les typons en dehors de toute fonction, pour en modifier ensuite les valeurs avec les données du XML importé. Dans ce cas, nous pouvons exploiter les valeurs modifiées par le XML depuis toute autre fonction présente dans notre programme.

Dans cet ouvrage, nous abordons la première méthode. Pour la seconde, nous vous invitons à consulter des publications plus spécialisées sur l'interfaçage dynamique, comme le livre très pointu de Thibault Imbert (éd. Pearson) ou l'ouvrage plus accessible d'Anne Tasso (éd. Eyrolles). Nous fournissons les références complètes de ces deux livres en fin d'ouvrage.

Dans ce document, nous chargeons le fichier XML relatif à la galerie, qui comporte la structure suivante :

```

<?xml version = '1.0' encoding="utf-8" ?>
<galerie>
    <photo>
        <titre>GUERANDE</titre>
        <legende>Sol craquelé</legende>
        <image>images/galerie/photo0.jpg</image>
        <vignette>images/galerie/vignettes/photo0-vignette.jpg</vignette>
    </photo>
    <photo>
        <titre>GUERANDE</titre>
        <legende>Gorgue, canal amenant l'eau chargée en sel dans les tables salantes
        ➡ </legende>
        <image>images/galerie/photo1.jpg</image>
        <vignette>images/galerie/vignettes/photo1-vignette.jpg</vignette>
    </photo>
    <photo>
        <titre>GUERANDE</titre>
        <legende>Amas de sel</legende>
        <image>images/galerie/photo2.jpg</image>
        <vignette>images/galerie/vignettes/photo2-vignette.jpg</vignette>
    </photo>

```

```

<photo>
  <titre>GUERANDE</titre>
  <legende>Cristallisoir</legende>
  <image>images/galerie/photo3.jpg</image>
  <vignette>images/galerie/vignettes/photo3-vignette.jpg</vignette>
</photo>
<photo>
  <titre>GUERANDE</titre>
  <legende>Saliculture</legende>
  <image>images/galerie/photo4.jpg</image>
  <vignette>images/galerie/vignettes/photo4-vignette.jpg</vignette>
</photo>
<photo>
  <titre>GUERANDE</titre>
  <legende>Ramassage du sel</legende>
  <image>images/galerie/photo5.jpg</image>
  <vignette>images/galerie/vignettes/photo5-vignette.jpg</vignette>
</photo>
<photo>
  <titre>GUERANDE</titre>
  <legende>Outils du paludier</legende>
  <image>images/galerie/photo6.jpg</image>
  <vignette>images/galerie/vignettes/photo6-vignette.jpg</vignette>
</photo>
</galerie>

```

Ce document comporte une structure classique composée d'un nœud principal appelé ici galerie. Cette structure affiche autant de nœuds nommés photo qu'il y a d'images à intégrer dans notre galerie. Chaque nœud photo, à son tour, dispose d'entrées respectivement nommées titre, légende, image et vignette, véhiculant chacune une valeur que nous distribuons, avec ActionScript, dans les objets mis en scène dans le document Flash. Dans cette section, nous utilisons uniquement les trois premiers éléments de chaque nœud. Le quatrième, vignette, est déployée dans la section suivante.



Créer un fichier XML. Pour créer un fichier XML, utilisez n'importe quel éditeur de texte et enregistrez le code au format texte avec l'extension *xml*. Puis, dans la première ligne, spécifiez un encodage de type UTF-8 pour éviter les problèmes d'accents mal interprétés si vous éditez depuis un Macintosh : sur Mac, les fichiers texte sont nativement codés en ISO Mac. Vous devez donc enregistrer le texte avec une option qui permette d'exporter en UTF-8. L'instruction ajoutée dans le code affiche donc :

```
<?xml version = '1.0' encoding="utf-8" ?>
```

Dans la suite Adobe, Dreamweaver permet de réaliser facilement des documents XML clairement indentés et propose des assistants à la saisie. Pour plus d'informations sur la structure d'un fichier XML, consultez les ouvrages de Howard Goldberg ou Florent Nolot aux éditions Pearson.

Dans le Flash, depuis la fenêtre Actions, nous activons le chargement d'une première image dans la zone cible, comme vu précédemment :

```

//----- Chargement initial PHOTO AGRANDIE
// CHARGEMENT
var chargeurPhoto:Loader= new Loader();

```

```

var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
chargeurPhoto.load(cheminPhoto);

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➤ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
    chargement_mc.visible=true;
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded /
➤ evt.currentTarget.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET1);
function chargementCOMPLET1(evt:Event) {
    cible_mc.addChild(chargeurPhoto);
    chargement_mc.visible=false;
    cible_mc.x=stage.stageWidth/2-cible_mc.width/2;
    cible_mc.y=stage.stageHeight/2-cible_mc.height/2-25;
}

```

Ensuite, nous activons la gestion du flux XML à partir duquel nous allons définir les contrôles de navigation sur les boutons suivant et retour :

```

//----- Chargement du XML
var i:Number=0;
var ecartEntreVignettes:Number=90;

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLCompleet);

function XMLCompleet(evt:Event) {
    // gestion des données XML
}

```

Avant de charger le fichier XML, nous initialisons un nombre *i*, qui permet de mémoriser l'ordre d'affichage de l'image courante. Cette valeur est initialisée à 0, ce qui correspond de nouveau à l'image chargée par défaut, "photo0.jpg".

Puis, nous importons le fichier XML. Pour ce faire, nous déclarons d'abord un nouveau chargeur (chargeurXML), en utilisant cette fois une instance de la classe URLLoader, comme nous le ferions pour importer une image. Ce chargeur active le chargement d'un fichier défini à travers l'objet URLRequest, passé directement en paramètre de la méthode load(). Cela équivaut à définir l'URL séparément avec `var monChemin:URLRequest=new URLRequest("monURL")`. Les deux méthodes conviennent.



Différence entre Loader et URLLoader. La classe Loader est utilisée pour charger un fichier image ou SWF. Tandis que la classe URLLoader est employée pour charger un fichier au format texte. La classe URLLoader est donc la méthode qu'il faut utiliser pour la gestion de données dynamiques.

Le chargeur chargeurXML est associé à un écouteur (avec l'action `addEventListener`) qui, à travers la classe Event et l'événement `COMPLETE`, appelle la fonction nommée `XMLComple` une fois le chargement effectué.

À l'intérieur de cette fonction, nous créons une variable `donneesXML` qui enregistre l'ensemble de l'arbre XML importé. Cela permettra, par la suite, de s'y référer aisément en invoquant cet objet pour distribuer chaque donnée qu'il véhicule, plus loin dans le code.

L'objet créé est un objet de type XML. Il se réfère à la classe Event active, via l'identifiant `evt`, et y cible les données courantes (`target.data`), données alors aspirées par l'écouteur auquel cette classe se rattache.

```
function XMLComple(evt:Event) {
    var donneesXML:XML=new XML(evt.target.data);
    var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();
}
```

Enfin, nous déclarons une première valeur à partir de l'objet XML que nous venons de créer, pour identifier le nombre de nœuds disponibles dans ce fichier. Pour cela, nous nous référons à l'objet XML avec `donneesXML`. Puis, nous spécifions le nom du nœud de premier niveau (`photo`) et détectons la longueur (la quantité de nœuds) présente dans le document, grâce à la méthode `length()`.



Lire le XML. Pour lire les données contenues dans un fichier XML, selon le type de données relevées, procédez comme suit :

- Pour atteindre un nœud précis, dont vous connaissez l'ordre d'apparition, passez son numéro entre crochets, comme ceci : `donneesXML.photo[numeroDuNoeud].toString()`. L'ordre d'affichage des nœuds dans l'arborescence du fichier XML démarre toujours à la valeur 0. Pour atteindre le premier nœud, nous inscrivons `nomDuXML.nomDuNoeuds[0]`.
- Si la structure comporte plusieurs nœuds imbriqués, ciblez chaque niveau individuellement, comme ceci : `donneesXML.photo[numeroDuNoeud].image[numeroDuNoeudDeSecondNiveau].toString()` ;
- Si le nœud porte la structure suivante : `<photo largeur="600" hauteur="450"> <image>images/photo0.jpg</image> </photo>`, pour cibler la valeur contenue dans l'attribut `largeur`, utilisez la syntaxe `donneesXML.photo[numeroDuNoeud].@largeur.toString()`.
- Si le nœud porte cette structure-ci : `<photo largeur="600" hauteur="450"> <image>images/photo0.jpg</image> </photo>`, pour cibler la valeur contenue entre les balises `<image>` et `</image>`, utilisez la syntaxe `donneesXML.photo[numeroDuNoeud].image.toString()`. Attention toutefois, toute XMLListe se comporte comme un XML si elle ne contient qu'un seul élément. Dans le cas contraire, il faudrait écrire `donneesXML.photo[numeroDuNoeud].image[0].toString()`.
- Si le nœud doit comporter des balises HTML, enveloppez le HTML dans une description de type `CDATA` : `<photo> <legende> <![CDATA[Marais salants de <b>GUERANDE</b>`

b>]]> </legende> </photo>, et ciblez la valeur contenue entre les balises <legende > et </legende>, avec la syntaxe `donneesXML.photo[numeroDuNoeud].legende.toString()`.

- Pour cibler enfin plus spécifiquement un nœud dont vous connaissez la valeur d'un attribut : <photo largeur="600" hauteur="450"> <image>images/photo0.jpg</image> </photo>, utilisez la syntaxe suivante : `donneesXML.photo[numeroDuNoeud].(@largeur=="600").toString()`.
- Pour enfin convertir le contenu d'une balise en chaîne de caractères, spécifiez `.toString()`, qui demeure facultatif.

Une fois ces informations définies, nous pouvons distribuer les données à travers les objets du document Flash :

```
// RETOUR
retour_btn.addEventListener(MouseEvent.CLICK, retourDOWN);
function retourDOWN(evt:MouseEvent) {
    i--;
    if (i<0) {
        i=NombreDeNoeudsDansLeXML-1;
    }
    distribuerValeurs();
}
```

Dans la fonction `XMLCompleter`, nous définissons successivement les écouteurs et les fonctions rattachées aux boutons suivant et retour de l'interface, ceci afin de directement pouvoir y exploiter les valeurs du XML.

Dans la fonction `retourDOWN`, activée lorsque l'utilisateur clique sur le bouton `retour_btn`, nous commençons par réduire la valeur de `i`, initialement portée à 0 (`i--`). Si cette valeur est inférieure à 0 (`i<0`), une fois diminuée, elle est aussitôt ramenée à la valeur correspondant au seuil limite de la galerie, c'est-à-dire au nombre de nœuds présent dans le fichier XML, en l'occurrence 7 moins un (`i=nombreDeNoeudDansLeXML-1`).

À la fin de la fonction, nous appelons une autre fonction (`distribuerValeurs`) qui rassemble les instructions communes aux deux boutons, comme vu précédemment.

Nous déclinons ensuite l'écouteur et la fonction pour le bouton `suivant_btn`, avant de refermer la fonction principale associée au flux XML :

```
// SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
function suivant(evt:MouseEvent) {
    i++;
    if (i==NombreDeNoeudsDansLeXML) {
        i=0;
    }
    distribuerValeurs();
}
```

Pour le bouton `suivant_btn`, nous inversons les valeurs définies pour la condition `if`, en spécifiant que la valeur de `i` doit revenir à 0 si celle-ci atteint la valeur correspondant au nombre de nœuds dans le XML (donc, lorsque `i` vaut 7). Ainsi, nous passons directement de la valeur 6 à 0, ce qui nous permet de créer un effet de boucle sur le déroulement des images pour notre galerie.

Enfin, la fonction `distribuerValeurs` qui termine le programme rassemble les instructions communes pour les deux boutons :

```
function distribuerValeurs () {  
    titre_txt.htmlText=donneesXML.photo[i].titre.toString();// titre  
    legende_txt.htmlText=donneesXML.photo[i].legende.toString();// légende  
    cheminPhoto=new URLRequest(donneesXML.photo[i].image.toString());// image agrandie  
    chargeurPhoto.load(cheminPhoto);  
}
```

Une fois la gestion du nombre `i` assurée, il reste à passer les valeurs contenues dans les éléments du XML vers les objets de l'interface Flash. Pour chacune des trois instructions qui ponctuent la fonction, nous faisons référence à des nœuds de l'objet `donneesXML`, en y ciblant plus spécifiquement les balises `titre`, `legende` et `image`. Pour convertir toutefois les valeurs en chaîne de caractères, et éviter d'importer les balises qui enveloppent notre sélection dans le fichier XML, nous utilisons la méthode `toString()`.

Vous remarquez que le chemin qui appelle les images est enregistré dans le document XML. Nous aurions très bien pu utiliser aussi une référence au fichier image en passant la valeur de `i` directement dans l'URL de l'image à importer, comme suit :

```
cheminPhoto=new URLRequest("images/galerie/photo"+i+".jpg");
```

Mais en appelant l'entrée contenue dans le XML, nous évitons de devoir publier à nouveau le document Flash si une modification devait être apportée dans l'organisation des fichiers. Nous permettons en outre de centraliser la gestion des données dans un seul document, le XML.

À retenir

- Pour optimiser la gestion des données dans un document Flash, il est plus confortable de les isoler dans un fichier XML. Cela évite de publier le document Flash à chaque modification des données.
- Les données isolées dans un fichier XML ne peuvent être distribuées que lorsque le XML est entièrement chargé. Il faut donc, soit les distribuer dans la fonction exécutée à l'issue du chargement du XML, soit les stocker dans des variables globales.
- Pour éviter que l'incrémement d'une variable nombre (`i`) ne risque de faire référence à un nœud inexistant, nous pouvons interrompre ou réinitialiser l'incrémement à l'aide d'une simple instruction `if` conditionnelle.
- Nous ciblons le nœud d'un document XML en nous référant à l'arbre racine préalablement identifié, puis en ciblant chacun des nœuds descendant de l'arborescence du fichier XML avec la méthode pointée. Il est également possible de cibler ponctuellement un attribut et de vérifier sa valeur. Vous pouvez enfin atteindre un nœud en fonction de son ordre d'apparition dans l'arborescence XML avec une valeur passée en paramètre du nœud ciblé, entre crochets comme `[i]`.

Interactivité sur les objets dynamiques

Nous savons comment associer des actions sur un objet placé sur la scène, mais lorsque l'objet ou les instructions traitent des éléments gérés dynamiquement, il faut savoir identifier ces objets et ces données afin de permettre toute interaction avec eux.

Dans cette section, nous présentons une déclinaison de notre galerie en plaçant dynamiquement des vignettes dans une zone d'affichage constituée par un symbole de type MovieClip. En cliquant sur les vignettes générées dynamiquement, une action effectue un zoom sur un autre symbole et y charge l'image agrandie, correspondant à la vignette activée. Les champs de texte dynamiques sont mis à jour également à chaque action (voir Figure 5.13).

Figure 5.13

Aperçu de la galerie à la publication.



Exemples > ch5_galleriesImages_5 fla

Dans la scène principale du document "ch5_galleriesImages_5 fla", au-dessus du calque `fond_mc`, le symbole `cible_mc` sert de conteneur pour les images agrandies que nous allons charger en cliquant sur les vignettes du bas. Ce conteneur contient aussi un autre symbole, `carreBlanc_mc`. Ce symbole est masqué par défaut avec la propriété `visible` passée sur `false` via ActionScript. Ce carré blanc, lorsqu'il est visible, sert de signalétique pour permettre à l'utilisateur d'identifier l'objet cliqué. À chaque clic, nous prévoyons de le redimensionner et de le repositionner sous l'objet cliqué pour le souligner et reconstituer dynamiquement un effet cliqué.

Les autres éléments sont similaires à ceux des sections précédentes. Nous retrouvons la jauge de chargement, les boutons suivant et retour, les champs de texte dynamiques, ainsi qu'un calque qui affiche des éléments graphiques de l'interface nommé `interface`. Nous

avons ajouté ici, au-dessus des autres calques, un symbole `cibleVignettes_mc`, vide, et un masque, pour y placer les vignettes qui sont gérées dynamiquement. Le masque sert à restreindre la zone d'affichage de ce calque pendant son déplacement, lorsque nous cliquons sur les boutons suivant et retour (voir Figures 5.14 et 5.15).

Figure 5.14

Aperçu de la scène principale.

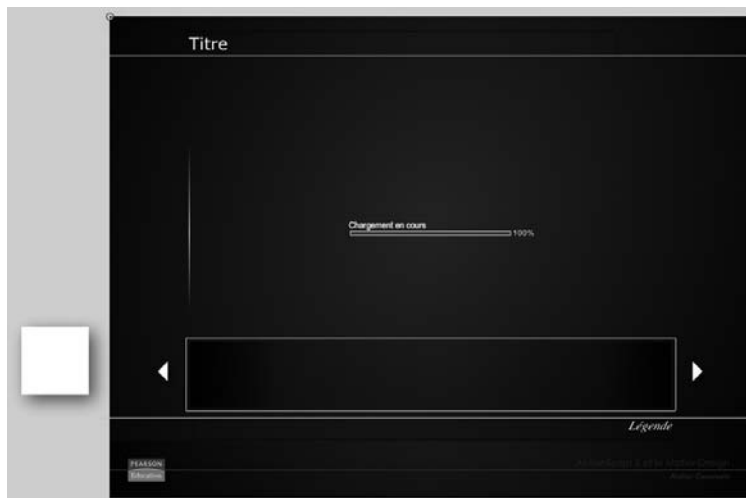
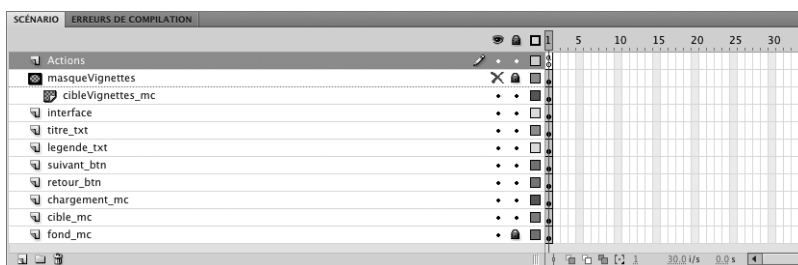


Figure 5.15

Aperçu du scénario de la scène principale.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

var i:int=0;
var ecartEntreVignettes:Number=90;
var positionCibleVignettesInit:Number=cibleVignettes_mc.x;
cibleVignettes_mc.carreBlanc_mc.x=-2;
cibleVignettes_mc.carreBlanc_mc.y=-2;
cibleVignettes_mc.carreBlanc_mc.visible=false;

//----- Chargement initial PHOTO AGRANDIE
```

```

// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest;

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➤ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
    chargement_mc.visible=true;
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current-
➤ Target.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurPhoto);
    chargement_mc.visible=false;
}

//----- Chargement du XML

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLCompleT);

function XMLCompleT(evt:Event) {
    var donneesXML:XML=XML(evt.target.data);
    var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();

    for (i; i<NombreDeNoeudsDansLeXML; i++) {
        // chargement des vignettes
        var chargeurVignette:Loader= new Loader();
        var vignette:String=donneesXML.photo[i].vignette.toString();
        var cheminVignette:URLRequest=new URLRequest(vignette);
        chargeurVignette.load(cheminVignette);
        chargeurVignette.x=ecartEntreVignettes*i;
        chargeurVignette.name=String(i);
        cibleVignettes_mc.addChild(chargeurVignette);

        // actions sur vignettes
        cibleVignettes_mc.addEventListener(MouseEvent.CLICK,clickVignettes);
        function clickVignettes(evt:MouseEvent) {
            cibleVignettes_mc.carreBlanc_mc.visible=true;
            titre_txt.htmlText=donneesXML.photo[evt.target.name].titre.toString();

            legende_txt.htmlText=donneesXML.photo[evt.target.name].legende.toString();
            cheminPhoto=new URLRequest("images/galerie/photo"+evt.target.name+".jpg");
            chargeurPhoto.load(cheminPhoto);
            TweenMax.to(cibleVignettes_mc.carreBlanc_mc, 1,
➤ {x:(ecartEntreVignettes*(evt.target.name-1))-2, ease:Strong.easeOut});
        }
        // transition carreBlanc
        cible_mc.scaleX=0.1;
        cible_mc.scaleY=0.1;
        cible_mc.x=mouseX;
        cible_mc.y=mouseY;
    }
}

```

```

        TweenMax.to(cible_mc, 3, {x:100, y:50, scaleX:1, scaleY:1, ease:Strong.easeOut});
    }
}
}

//----- Boutons NAVIGATION

retour_btn.addEventListener(MouseEvent.CLICK, retourDOWN);
function retourDOWN(evt:MouseEvent) {
    if (cibleVignettes_mc.x<=positionCibleVignettesInit*2) {
        TweenMax.to(cibleVignettes_mc, 1,
        ➤ {x:cibleVignettes_mc.x+ecartEntreVignettes, ease:Elastic.easeOut});
    }
}
suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
function suivant(evt:MouseEvent) {
    if (cibleVignettes_mc.x>=0) {
        TweenMax.to(cibleVignettes_mc, 1, {x:cibleVignettes_mc.x-ecartEntre-
        ➤ Vignettes, ease:Elastic.easeOut});
    }
}
}

```

Le code est structuré en quatre étapes. Il fonctionne de la manière suivante : nous commençons par importer les classes nécessaires et nousinstancions le module de chargement d'images, sans activer de chargement. Ainsi, nous plaçons les fonctions liées au chargement et à la jauge de progression indépendamment de la gestion des données XML. Dans un deuxième temps, nous traitons les données importées par le XML, à travers la fonction XMLCompleet. Nous permettons ainsi de charger les vignettes dynamiquement et les distribuer dans notre interface avec un positionnement précis. Dans un troisième temps, dans la fonction XMLCompleet, nous définissons les actions au clic sur les vignettes. Nous terminons notre développement en spécifiant quelques actions, indépendantes des données XML, pour contrôler le déplacement du conteneur de vignettes dans notre dispositif global de navigation. Il n'y a pas de relation entre les données XML importées et ce dispositif de navigation si ce n'est que la répercussion indirecte de la quantité de vignettes chargées dans le conteneur, qui aidera à définir les limites de son positionnement.

Tout d'abord, nous commençons par appeler les classes requises pour les animations TweenMax :

```

//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

```

Puis, nous initialisons un certain nombre de variables :

```

var i:Number=0;
var ecartEntreVignettes:Number=90;
var positionCibleVignettesInit:Number=cibleVignettes_mc.x;
cibleVignettes_mc.carreBlanc_mc.x=-2;
cibleVignettes_mc.carreBlanc_mc.y=-2;
cibleVignettes_mc.carreBlanc_mc.visible=false;

```

La première, le nombre `i`, permet de définir l'index (la position courante) de l'image active, chargée dans la zone d'agrandissement `cible_mc`.

La variable `ecartEntreVignettes` sert à déterminer en une seule fois la valeur qui sépare chaque point d'origine de chaque vignette que le script va placer dans `cibleVignettes_mc`. Cette valeur (90) est utilisée à plusieurs reprises. Pour simplifier la gestion du code, nous la véhiculons à travers une variable.

La variable `positionCibleVignetteInit` sert à mémoriser la position de départ du conteneur de vignettes pour être exploitée plus tard dans le calcul des limites du défilement du conteneur de vignettes `cibleVignettes_mc`.

Puis, les trois instructions suivantes affectent le symbole `carreBlanc_mc`, placé dans le symbole `cibleVignettes_mc` :

```
cibleVignettes_mc.carreBlanc_mc.x=-2;
cibleVignettes_mc.carreBlanc_mc.y=-2;
cibleVignettes_mc.carreBlanc_mc.visible=false;
```

Nous déterminons la position de départ du carré blanc en le calant en X et en Y à 2 pixels plus haut et à gauche de l'origine du clip qui le contient (celle du symbole `cibleVignettes_mc`), car le carré blanc mesure 4 pixels de large de plus que les vignettes. En le décalant de 2 pixels en haut et à gauche, nous centrons ce carré blanc. Nous formalisons ainsi un contour signalétique de 2 pixels qui souligne chaque vignette cliquée. À chaque clic de souris sur l'une d'entre elles, l'objet est repositionné sous la vignette. Par défaut, le carré blanc est rendu invisible (`visible=false`), car aucune image n'est encore affichée dans la zone d'agrandissement. Mais il est de nouveau visible en cliquant sur l'une ou l'autre des vignettes.

Puis, nous mettons en place le chargement d'images et la jauge de progression, sans activer toutefois de chargement sur une première image ni l'ajouter sur la scène :

```
//----- Chargement initial PHOTO AGRANDIE

// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest;

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➤ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
    chargement_mc.visible=true;
    var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current-
➤ Target.bytesTotal);
    chargement_mc.barre_mc.scaleX=valeurPourcentage;
    chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+ "%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,
➤ chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
    cible_mc.addChild(chargeurPhoto);
    ➤ chargement_mc.visible=false;
}
```

Une fois les fonctions de gestion du chargement définies, nous importons le document XML, avec le chargeur `chargeurXML`, la méthode `load` qui active le chargement du contenu, et l'écouteur qui appelle la fonction une fois le chargement effectué. Dans cette fonction, nous retrouvons la définition de l'arbre XML véhiculé par la variable `donneesXML` et la longueur de l'arbre, désignée par `nombreDeNoeudsDansLeXML` :

```
//----- Chargement du XML

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLCompleet);

function XMLCompleet(evt:Event) {
    var donneesXML:XML=XML(evt.target.data);
    var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();
}
```

À la suite de la définition des premières variables, nous pouvons identifier l'utilisation d'une boucle `for` :

```
for (i; i<NombreDeNoeudsDansLeXML; i++) {
    // actions à répéter autant de fois que d'itérations induites par la boucle.
}
```

Une boucle `for` permet de rassembler, dans un seul constructeur, un ensemble d'instructions répétitives, dont seul un ou quelques paramètres changent. En l'occurrence, seule la valeur véhiculée par le nombre `i` change pour charger, positionner et définir des actions et interactions avec les vignettes.

Dans cette boucle, nous indiquons de considérer la valeur de `i` telle que définie initialement (`i` vaut 0 tel que nous l'avons défini en amont), puis, si `i` reste inférieur au nombre de nœuds présents dans le XML (`i<nombreDeNoeudsDansLeXML`), alors, nous l'incrémentons (`i++`).



Mécanisme d'une boucle `for`. Une boucle `for` présente la structure suivante :

```
for(valeur de référence ; condition à respecter ; modification de la valeur de
référence){

    // instructions à répéter autant de fois que d'itérations dans la boucle.
}
```

À l'intérieur de la boucle `for`, nous plaçons les instructions à reproduire :

```
// chargement des vignettes
var chargeurVignette:Loader= new Loader();
var vignette:String=donneesXML.photo[i].vignette.toString();
var cheminVignette:URLRequest=new URLRequest(vignette);
chargeurVignette.load(cheminVignette);
chargeurVignette.x=ecartEntreVignettes*i;
chargeurVignette.name=String(i);
cibleVignettes_mc.addChild(chargeurVignette);

// actions sur vignettes
cibleVignettes_mc.addEventListener(MouseEvent.CLICK,clickVignettes);
function clickVignettes(evt:MouseEvent) {
```



```

cibleVignettes_mc.carreBlanc_mc.visible=true;
titre_txt.htmlText=donneesXML.photo[evt.target.name].titre.toString();
legende_txt.htmlText=donneesXML.photo[evt.target.name].legende.toString();
cheminPhoto=new URLRequest("images/galerie/photo"+evt.target.name+".jpg");
chargeurPhoto.load(cheminPhoto);
TweenMax.to(cibleVignettes_mc.carreBlanc_mc, 1, {x:(ecartEntreVignettes*
    (evt.target.name-1))-2, ease:Strong.easeOut});
// transition carreBlanc
cible_mc.scaleX=0.1;
cible_mc.scaleY=0.1;
cible_mc.x=mouseX;
cible_mc.y=mouseY;
TweenMax.to(cible_mc, 3, {x:100, y:50, scaleX:1, scaleY:1, ease:Strong.easeOut});
}

```

La première série d'instructions à reproduire définit le chargement des vignettes et utilise, en paramètre de l'URL, la valeur incrémentée de `i`. Ainsi, à chaque itération, chaque image définie dans le XML sera chargée successivement jusqu'à la dernière, stoppée par la condition de la boucle `for` (`i<nombreDeNoeudsDansLeXML`) :

```

// chargement des vignettes
var chargeurVignette:Loader= new Loader();
var vignette:String=donneesXML.photo[i].vignette.toString();
var cheminVignette:URLRequest=new URLRequest(vignette);
chargeurVignette.load(cheminVignette);
chargeurVignette.x=ecartEntreVignettes*i;
chargeurVignette.name=String(i);
cibleVignettes_mc.addChild(chargeurVignette);

```

Pour permettre, par la suite, d'attacher un écouteur à chaque objet placé dynamiquement, nous devons pouvoir identifier chaque objet individuellement. Pour ce faire, nous utilisons la propriété `name`, que nous attachons à chaque objet chargé (`chargeurVignette.name=String(i)`) en affectant, pour valeur de nom, la valeur correspondant à l'ordre d'affichage de chacune de ces vignettes. Ainsi, la vignette 0 se nomme 0, la vignette 1 se nomme 1, et ainsi de suite. Pour cibler chaque objet plus tard, il suffira d'invoquer son nom en utilisant de nouveau la propriété `name`.

Dans ces instructions de chargement des vignettes, nous relevons aussi le positionnement dynamique est également orchestré par la propriété `x`. Nous déterminons cette valeur en multipliant l'écart défini plus haut entre les vignettes (90 pixels) par la valeur de `i` avec `chargeurVignette.x=ecartEntreVignettes*i`. Donc, chaque vignette ajoutée à toute nouvelle itération est placée à 90 pixels de plus que la précédente. Comme chacune d'entre elles mesure précisément 80 pixels de large. Nous obtenons une marge de 10 pixels entre chaque vignette.

Une fois les vignettes affichées, nous ajoutons un écouteur sur l'objet conteneur global `cibleVignettes_mc`. Pour connaître l'objet sur lequel l'utilisateur a cliqué, nous le ciblons grâce à la propriété `name` associée à l'expression `evt.target`. Cette propriété permet d'identifier un objet lorsqu'il est cliqué au sein du conteneur auquel est rattaché l'écouteur :

```

cibleVignettes_mc.addEventListener(MouseEvent.CLICK,clickVignettes);
function clickVignettes(evt:MouseEvent) {
    // actions à exécuter
    trace (evt.target.name)
}

```

Différence entre target et currentTarget

Dans l'expression `evt.target.name`, nous désignons la propriété `name` (le nom) de l'élément activé contenu dans l'objet (`target`) auquel est rattaché la classe invoquée par l'écouteur, et donc, l'événement qui lui est associé (`evt`). Cette expression (`target`), est employée lorsque plusieurs objets isolés dans un même et unique conteneur exécutent la même fonction.

Dans l'expression `evt.currentTarget.name`, nous désignons la propriété `name` (le nom) de l'objet attaché à l'écouteur lui-même (`evt.currentTarget`). Cette expression (`currentTarget`) est utilisée lorsque plusieurs objets isolés et sans relation exécutent la même fonction.

Considérons l'exemple suivant : prenons deux ensembles, un ensemble de lettres et un ensemble de chiffres. Plaçons un écouteur sur chacun d'eux. Les deux ensembles invoquant la fonction alors (`evt : Event`), la propriété `target` renverra la lettre ou le chiffre qui aura intercepté le clic, tandis que `currentTarget` renverra le groupe auquel l'écouteur est appliqué et dont l'événement a été intercepté.

Ainsi, il n'est pas nécessaire de connaître à l'avance l'objet qui va être activé et qui va exécuter la fonction pour lui affecter un comportement.

Dans cette fonction, nous remplaçons l'action `trace`, citée en exemple, par les instructions suivantes :

```
cibleVignettes_mc.carreBlanc_mc.visible=true;
titre_txt.htmlText=donneesXML.photo[evt.target.name].titre.toString();
legende_txt.htmlText=donneesXML.photo[evt.target.name].legende.toString();
cheminPhoto=new
➤ URLRequest(donneesXML.photo[evt.target.name].image.toString());
chargeurPhoto.load(cheminPhoto);
TweenMax.to(cibleVignettes_mc.carreBlanc_mc, 1,
➤ {x:(ecartEntreVignettes*(evt.target.name-1))-2, ease:Strong.easeOut});
cible_mc.scaleX=0.1;
cible_mc.scaleY=0.1;
cible_mc.x=mouseX;
cible_mc.y=mouseY;
TweenMax.to(cible_mc, 3, {x:100, y:50, scaleX:1, scaleY:1,
➤ ease:Strong.easeOut}); // transition Zoom
```

À chaque clic sur une vignette, nous réactivons l'affichage du carré blanc, masqué par défaut, avec la propriété `visible` passée sur `true` :

```
cibleVignettes_mc.carreBlanc_mc.visible=true;
```

Puis, nous modifions le texte contenu dans les champs de texte dynamiques `titre_txt` et `legende_txt` :

```
titre_txt.htmlText=donneesXML.photo[evt.target.name].titre.toString();
```

Pour déterminer le nœud actif, nous utilisons un nœud de l'arbre XML `donneesXML` (`donneesXML.photo`) et spécifions le nom de l'objet cliqué avec `evt.target.name`, entre crochets.

Nous procédons de même pour désigner le nœud correspondant à l'URL de l'image agrandie :

```
legende_txt.htmlText=donneesXML.photo[evt.target.name].legende.toString();
```

Le chargeur relance ensuite l'affichage de l'image agrandie :

```
chargeurPhoto.load(cheminPhoto);
```

Enfin, nous appliquons une transition TweenMax sur la position horizontale du carré blanc pour créer ainsi l'effet de déplacement. Seul le paramètre x doit être calculé et défini. Il suffit de récupérer au travers de son nom l'index de position de la vignette et de le multiplier par l'espacement existant entre chaque vignette (90 pixels) puis de retrancher le décalage de 2 pixels nécessaire pour assurer la marge du carré blanc.

Attention, comme lors du chargement la première vignette est déjà en place, il nous faut en tenir compte et démarrer l'incréméntation à 1, donc retrancher 1 à la valeur d'index de la vignette. À défaut de cette correction, nous observerions un décalage de 90 pixels vers la droite.

Nous terminons enfin le code par la définition d'actions sur les boutons suivant et retour, pour modifier la position du conteneur cibleVignettes_mc, à chaque clic sur les flèches correspondantes :

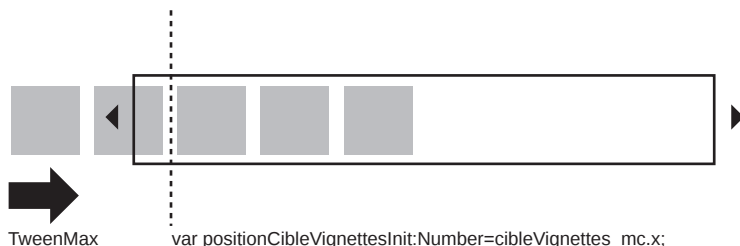
```
//----- Boutons NAVIGATION

retour_btn.addEventListener(MouseEvent.CLICK, retourDOWN);
function retourDOWN(evt:MouseEvent) {
    if (cibleVignettes_mc.x<=positionCibleVignettesInit*2) {
        TweenMax.to(cibleVignettes_mc, 1,
        {x:cibleVignettes_mc.x+ecratEntreVignettes, ease:Elastic.easeOut});
    }
}
suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
function suivant(evt:MouseEvent) {
    if (cibleVignettes_mc.x>=0) {
        TweenMax.to(cibleVignettes_mc, 1, {x:cibleVignettes_mc.x-ecratEntreVignettes,
        ➡ ease:Elastic.easeOut});
    }
}
```

Dans chacune de ces deux fonctions, l'animation TweenMax est activée uniquement si la position de la cible des vignettes l'y autorise. Plus exactement, l'animation est lancée si, pour le déplacement du menu dans le sens du retour, sa position courante reste inférieure à sa position initiale (voir Figure 5.16). Nous multiplions cependant la valeur par deux afin de recentrer arbitrairement l'objet dans la scène. Mais ce n'est pas une obligation.

Figure 5.16

Schéma du repositionnement du conteneur de vignettes.



Nous procédons au même type de calcul dans l'autre sens, avec la deuxième fonction. Nous spécifions alors que l'animation TweenMax peut se produire tant que la position de l'objet ne passe pas par la limite de gauche qui vaut 0.

À retenir

- Pour gérer une construction dynamique d'interface dont les mécanismes de mise en forme se répètent, nous pouvons utiliser une boucle `for`.
- Pour associer une action à un objet généré dynamiquement, il est possible de cibler l'objet en utilisant sa propriété `name`.
- La propriété `target` se distingue de la propriété `currentTarget` en cela qu'elle cible l'objet activé, contenu dans l'objet attaché à l'écouteur, alors que `currentTarget` cible uniquement l'objet attaché à l'écouteur. La propriété `currentTarget` est donc employée pour distinguer plusieurs objets isolés qui exécutent la même fonction. La propriété `target` est utilisée pour désigner plusieurs objets d'un même conteneur qui exécutent la même fonction.

Résoudre les erreurs éventuelles au chargement et à l'exécution

Des erreurs de chargement ou d'exécution peuvent apparaître lorsque vous produisez avec des fichiers externalisés (pour le chargement) ou lorsque vos développements sont lourds ou exécutés sur des systèmes instables ou à bout de souffle (pour l'exécution). Une erreur, fût-elle bénigne, peut, dans certains cas, neutraliser l'ensemble de votre application si celle-ci n'est pas orientée vers une action alternative.

Dans cette section, nous présentons les instructions qui permettent d'identifier les erreurs et celles qui permettent de les résoudre, à travers des actions `trace`.



Exemples > `ch5_galleriesImages_6 fla`

Dans la scène principale du document "`ch5_galleriesImages_6 fla`", au-dessus du calque `fond_mc`, figure un unique calque actions (voir Figure 5.17).

Figure 5.17

Aperçu de la fenêtre de scénario de la scène principale.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
//----- Chargement du XML
var chargeurXML:URLLoader = new URLLoader();
```

```
//chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.load(new URLRequest("xml/g.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLComplet);

chargeurXML.addEventListener(IOErrorEvent.IO_ERROR, siErreur);
function siErreur(evt:IOErrorEvent) {
    trace("Erreur de chargement");
}

function XMLComplet(evt:Event) {
    trace("Document XML chargé");
    //
    try {
        trace("Document correctement exécuté");
    }
    catch (monErreur:Error) {
        trace("Document mal exécuté. Voici l'erreur = "+monErreur);
    }
    finally {
        trace("Instruction post analyse. Exécutée avec ou sans erreur, après l'analyse.");
    }
}
```

Dans notre document, nous chargeons le fichier XML utilisé pour la réalisation de la galerie photo. Mais aucune action n'est distribuée pour traiter les données. Nous avons simplement placé dans le code les instructions qui permettent, à chaque étape, de localiser tout éventuel problème de chargement ou d'exécution. Les actions trace qui apparaissent permettent alors d'identifier l'apparition de tel ou tel problème et, si besoin, de substituer ces actions trace par des instructions qui proposent un contenu alternatif à ce qui devait apparaître en l'absence d'erreur.

Ainsi, dans le code de ce document, nous avons mis en commentaire la ligne qui fait référence à l'URL du document XML valide. Puis, nous l'avons dupliquée en remplaçant le nom du fichier valide par un nom invalide. Ainsi, le fichier qui n'existe pas génèrera une erreur de chargement.

Pour traiter une erreur liée au chargement, nous plaçons un écouteur qui utilise l'événement `IO_ERROR`, disponible via la classe `IOErrorEvent`. Cela donne :

```
chargeurXML.addEventListener(IOErrorEvent.IO_ERROR, siErreur);
function siErreur(evt:IOErrorEvent) {
    trace("Erreur de chargement générée volontairement dans le cadre de cet exemple");
}
```

Lorsque le document est publié, le fichier inexistant n'est pas trouvé. Le message, inscrit dans l'action trace ("erreur de chargement générée volontairement dans le cadre de cet exemple"), apparaît aussitôt dans la fenêtre de sortie. En réactivant l'instruction `URLRequest` valide et en neutralisant la commande erronée, le message lié à l'erreur de chargement n'apparaît plus lorsque le document est à nouveau publié.

Plus loin, une fois le document XML chargé correctement, si l'instruction valide a été réactivée, nous pouvons tester les autres formes d'erreurs liées à l'exécution. Les commandes

qui gèrent ce type d'erreurs sont placées à l'intérieur de la fonction appelée à l'issue du chargement du document XML.

Dans un premier temps, nous vérifions que le document a effectivement bien été chargé. Pour cela, nous introduisons une simple action trace au début de la fonction. Cette action ne vérifie rien sinon qu'elle ne peut être exécutée que si le document a effectivement bien été chargé et elle nous informe donc sur le succès du chargement :

```
trace("Document XML chargé");
```

Ensuite, pour contrôler la validité des instructions qui seront exécutées après le chargement, toutes les lignes de code de cette fonction doivent être introduites dans le bloc d'instructions `try{}`. À défaut, les actions non incluses entre les deux accolades qui suivent la méthode `try` ne seront pas vérifiées. Pour tester la validité des actions contenues dans le bloc d'instructions `try{}`, nous y plaçons une action trace qui confirme la bonne exécution de l'ensemble du programme :

```
try {  
    trace("Document correctement exécuté");  
}
```

Un bloc `try` doit toujours être suivi, soit d'un bloc `catch{}`, soit d'un bloc `finally{}`, soit des deux.

Le bloc `catch{}` propose l'exécution d'une instruction alternative dans le cas où une erreur aurait été détectée. Par exemple, vous pouvez afficher à travers le bloc d'instruction `catch{}` un message. Ce message peut signaler, à l'utilisateur, une erreur d'exécution indépendante du site et lui suggérer de recharger le document ou de se reconnecter ultérieurement. Vous pouvez aussi exécuter une version allégée de votre programme proposé initialement dans le bloc `try{}`, mais cela augmente naturellement la charge de votre développement. Dans notre exemple, le bloc `catch{}` exécute une action trace qui affiche le type d'erreur rencontré, en invoquant la classe `Error` relative à cette commande :

```
catch (monErreur:Error) {  
    trace("Document mal exécuté. Voici l'erreur = "+monErreur);  
}
```

Le bloc `finally{}` s'exécute, lui, quel que soit le résultat obtenu par les blocs `try{}` et `catch{}`. Il sert à ponctuer l'exécution d'un programme par une action complémentaire. Ce bloc est moins usité que les deux précédents, mais comme son nom l'indique, il permet de finaliser un programme et orienter éventuellement l'utilisateur sur les actions à conduire. Il peut aussi servir à réinitialiser les valeurs initialisées lors de la tentative de la première méthode `try{}`. Dans notre exemple, une action trace termine le programme :

```
finally {  
    trace("Instruction post analyse. Exécutée si erreur ou sans erreur, après  
    l'analyse.");  
}
```

Mais, étant exécutée même lorsqu'il n'y a pas d'erreur, il convient d'employer le bloc `finally{}`, avec une structure conditionnelle qui initialise les valeurs, seulement si nécessaire.

À retenir

- Il est possible de gérer les erreurs de chargement grâce à la classe `IOErrorEvent`.
- Il est possible de gérer les erreurs d'exécution grâce à la méthode `try` et d'introduire une action alternative avec la méthode `catch`.
- Une méthode `finally` permet de ponctuer, à travers une structure conditionnelle, la détection d'une erreur par une instruction complémentaire, si nécessaire.

Gestion de site dynamique avec XML

Il se distingue deux types de développement : le développement front-office et le développement back-office.

Le développement front-office désigne la programmation effectuée sur un ordinateur local et exécutée dans le navigateur côté client. Même si ActionScript permet aussi de réaliser des développements côté serveur, ActionScript, JavaScript, HTML, sont des langages de développement utilisés généralement en front-office.

Le développement back-office consiste à programmer des applications qui sont exécutées côté serveur, et non plus localement. Si vous les réalisez localement, vous devez installer sur votre poste de travail un serveur d'émulation pour tester l'exécution de vos programmes. PHP, par exemple, est un langage de scripts très largement utilisé et exécuté côté serveur. Ce langage est en mesure de traiter des données depuis une base MySQL placée sur un serveur distant, et donc, de centraliser des données en vue d'être redistribuées dans un site. PHP peut également convertir assez simplement, les données stockées dans une base MySQL, en flux XML, comme si vous disposiez finalement d'un fichier XML localisé et stocké côté client.

Dès lors que vous savez exploiter XML dans la construction d'interfaces dynamiques en Flash, vous savez donc aussi appeler un même flux XML généré depuis un serveur distant, avec une application réalisée dans un langage dynamique tel que PHP, par un prestataire développeur back-office tiers, par exemple. Vous pouvez alors accéder, indirectement, et grâce aux instructions PHP, à des données centralisées dans une base MySQL, sous la forme d'une simple requête XML ou d'un simple lien vers un fichier PHP, que vous passez alors en paramètre de la méthode `URLRequest()` à l'intérieur de votre Flash (par exemple : `var chemin:URLRequest= new URLRequest("http://www.monsite.com/scriptQui-RenvoieUnFluxXML.php");`). Il devient donc relativement simple de confier la gestion des données à un développeur back-office et de vous concentrer sur votre propre développement en local en ActionScript, sans avoir à coder une seule ligne en PHP, sans jamais avoir à échanger le moindre fichier avec votre prestataire, ni installer quelque serveur que ce soit sur votre poste de travail, ni même placer vos documents Flash sur un serveur d'exécution pour les publier.

Il existe une autre alternative à la relation client/serveur, à base de Flash. C'est AMFPHP, qui permet à Flash de directement communiquer avec un serveur et inversement.

Cette méthode de répartition des tâches entre développeur front-office et back-office, grâce au XML, s'avère particulièrement efficace en production, car elle permet au graphiste codeur, de lui épargner la gestion d'un serveur ou d'un émulateur de serveur et au développeur back-office, de s'abstraire d'un document Flash parfois complexe à dépecer. Cette méthode permet, aussi, de bien séparer les tâches en programmation et de garantir que chaque partie concernée puisse se concentrer sur sa spécialité, sans jamais avoir à soumettre quelque partie que ce soit de son travail à un autre, tout en reliant pourtant dynamiquement les fichiers entre eux.

Synthèse

Dans ce chapitre, vous avez appris à optimiser considérablement le poids de vos documents lorsqu'ils affichent des images en grand format, en externalisant les contenus dans des répertoires placés au sein de votre site. Vous avez vu comment réaliser des galeries d'images et interagir avec des objets placés dynamiquement. Vous avez appris à centraliser des informations dans un document XML pour simplifier la maintenance de votre site. Enfin, vous savez maintenant contourner les erreurs éventuelles, observées au chargement ou à l'exécution des programmes. Vous êtes à présent en mesure de créer des interfaces dynamiques simples, et d'externaliser vos données dans un fichier XML. Vous êtes également au fait de la manière de traiter des données générées dynamiquement depuis une base placée sur un serveur distant si celles-ci sont distribuées sous la forme d'un lien ou d'une requête XML.

6

La vidéo standard et composite en FLV

Introduction

La gestion de la vidéo au sein de Flash est assez proche de celle de tout autre type de contenu. De cette manière, nous pouvons envisager son intégration indépendamment de son cadre habituel de diffusion, avec des dimensions libres et de l'interactivité, comme si nous traitons une image en somme.

Flash intègre la gestion de contenus vidéos depuis la version 6 (MX). Mais il compte aujourd'hui trois formats que nous regroupons dans deux grandes familles : le format FLV, flexible et universel, et le format F4V, plus "classieux" et récent. Le premier intègre deux codecs, le Sorenson Spark et le On2 VP6 tandis que le second propose uniquement le H-264. Nous abordons les spécificité du H-264 à travers le format F4V dans le chapitre suivant. Dans ce chapitre, nous détaillons l'encodage de la vidéo au format FLV avec ces deux premiers codecs, plus anciens.

Dans sa version 6, le lecteur Flash utilisait le codec Sorenson Spark, très compact, et a ainsi largement contribué à la diffusion à grande échelle de la vidéo sur le Web.

Avec Flash 8, l'apparition du codec On2 VP6 a permis d'améliorer la qualité des vidéos mouvementées dans Flash et de gérer la couche alpha éventuellement intégrée au flux vidéo.

Par vidéo standard, nous entendons donc l'ensemble des vidéos publiées au format FLV, encodée en Sorenson Spark, sans couche transparente, destiné à un affichage simple et compatible avec les plus anciennes versions du lecteur Flash (version 6).

Par vidéo composite, nous entendons la composition d'interfaces à partir de sources vidéos qui possèdent une couche transparente (couche alpha), encodée en on2VP6 avec la couche alpha. La transparence, lorsqu'elle est activée, permet de réaliser des interfaces graphiquement élaborées telles que l'incrustation d'un personnage filmé sur fond vert et détourné, des animations de particules, la superposition d'éléments de décor animés, des espaces publicitaires en premier plan d'une mise en page, etc.

Dans ce chapitre, nous détaillons d'abord les contraintes générales liées à la gestion d'une vidéo pour le Web. Nous abordons ensuite les techniques élémentaires de publication d'une vidéo depuis un logiciel tel que Apple Motion, disponible dans la suite Apple Final Cut Studio, ou Adobe After Effects de la suite vidéo Adobe, avec de la transparence. Nous voyons aussi l'encodage de vidéos standard vers Flash pour un export au format FLV, depuis tout type de logiciel. Enfin, nous traitons l'intégration simple de ces vidéos dans Flash, à l'aide du composant FLVPlayback.

À l'issue de ce chapitre sur la vidéo standard et composite au format FLV, vous serez en mesure de réaliser des interfaces contenant des vidéos simples, sophistiquées et optimisées pour toutes les versions de Flash qui acceptent la vidéo.

Concevoir la vidéo pour le Web

Lorsque vous préparez un contenu vidéo à destination du Web, vous devez considérer que sa diffusion doit obéir à quelques caractéristiques liées à la nature même du support de diffusion. Voici quelques questions/réponses qui vous guideront sur la création de contenus vidéos à destination du Web, et plus particulièrement, vers Flash.

Quel ratio de pixels ? Une vidéo échantillonnée pour le Web doit être définie en pixels carrés, de ratio 1,00. Les écrans d'ordinateur (qui exécutent la page web) possèdent une trame de pixels carrés, ce qui les distingue des écrans de télévision dont le ratio de pixels est respectivement 1,09 en PAL 4/3, 1,46 en PAL grand écran, 0,9 en NTSC 4/3 et 1,21 en NTSC grand écran.

Il est possible de convertir le ratio du format des vidéos à partir des logiciels Motion et After Effects. Pour connaître le ratio de vos vidéos, ouvrez-les dans l'une de ces applications et observez leurs propriétés. Pour adapter le document au format pixels carrés, définissez, dans les boîtes de dialogue qui apparaissent lors de la création d'une nouvelle séquence vidéo, un format de pixel carré et redimensionnez au besoin votre vidéo.

Avec ou sans trame ? La vidéo analogique est composée de deux images (trame paire et trame impaire). Dans une diffusion pour la télévision, il faut toujours synchroniser ces deux images afin de ne pas donner l'impression d'un tremblement. L'image numérique n'est composée, elle, que d'une seule trame par image, mais intégrale. Si vos images proviennent de sources analogiques (caméscopes, VHS et/ou autre support Hi8), vous devez convertir ces images en images dites progressives, c'est-à-dire sans trame. Si, à l'inverse, vos images proviennent de sources numériques, c'est inutile : elles sont déjà progressives.

Notez que les logiciels de montage et de trucage, ainsi que l'encodeur Adobe livré avec Flash, permettent de convertir d'un format vers l'autre.

Une capture progressive vous permettra aussi d'extraire plus facilement une image nette, non tramée, pour la traiter dans Photoshop, en vue éventuellement de l'intégrer dans une composition Flash ou vers un document imprimé.

Quelle cadence pour le Web ? La cadence (ou fréquence) d'images est le nombre d'images fixes affichées en une seconde de film (elle est à 25 images par seconde pour une vidéo standard en PAL). Si la cadence pour la télévision ou le cinéma est importante, elle peut être réduite pour le Web. Mais ceci n'a pas réellement d'impact sur le poids des fichiers, contrairement aux idées reçues. En effet, les algorithmes traitent essentiellement des images-clés par nature insensibles à une variation légère de la cadence des images, sauf pour des mouvements réellement très marqués. Dans ce cas, le nombre d'images-clés devient plus important et leur itération plus serrée sur l'échelle du temps. Une modification importante de la cadence peut sans doute, dans ce cas, alléger un peu le poids d'une vidéo.

Mais indépendamment du poids des vidéos, la cadence agit surtout sur les performances d'affichage de la carte graphique. Plus il y a de variations à gérer dans un même temps donné, plus le processeur vidéo est sollicité.

Si la vidéo est importée physiquement dans le scénario de Flash, sa cadence doit être identique à celle de la vidéo. En revanche, si la vidéo est stockée en dehors de l'animation Flash, et que celle-ci est affichée dynamiquement dans Flash à l'aide du composant FLVPlayback

ou avec les commandes de la classe `NetStream`, sa cadence peut être différenciée. Lorsqu'une vidéo est importée dans Flash, si la cadence est différente de celle du scénario, le son, les animations et les actions risquent en effet d'être désynchronisés par rapport à l'image.

Combien de vidéos en simultané ? Il est possible de multiplier les couches de fichiers vidéo au sein d'une même animation Flash, mais préférez les rassembler autant que possible dans un seul fichier vidéo. Vous gagnerez en fluidité.

Quelles dimensions pour une bonne image ? À débit équivalent, plus la vidéo occupe de la surface à l'écran, plus la compression devra être forte. Pour compenser le poids induit par les dimensions de la vidéo et stabiliser le débit, préférez réduire les dimensions des vidéos, vous obtiendrez un meilleur rendu. Dans Flash, les vidéos ne requièrent pas d'être importées dans un format standard de ratio 4/3 ou 16/9 par exemple. Les vidéos peuvent être recadrées avant l'encodage, de sorte que seule la surface utile à coder soit traitée sous la forme d'un signal vidéo utile. Par exemple, pour afficher un personnage qui marche en direction de l'utilisateur, une vidéo de forme rectangulaire verticale peut largement suffir. Le reste du décor sera traité en image fixe, réparti sur d'autres calques, dans Flash. Pour laisser voir le décor derrière le sujet filmé, il suffira de l'exporter avec une couche transparente (alpha) et de l'enregistrer au format FLV avec les options ON2 VP6 et canal alpha activées.

Images fixes ou en mouvement ? Plus la vidéo est riche d'un point de vue graphique (nous disons qu'elle possède du bruit), plus son poids sera conséquent, car c'est ce qui bouge qui est codé dans les algorithmes de compression. Ne rendez mobile que ce qui est utile ou alors, privilégiez les animations courtes.

Il est possible, dans certains logiciels de montage vidéo, de réduire le bruit sans trop altérer l'image. Étudiez cette option qui peut réduire considérablement la perte de qualité à l'encodage, ou, à qualité égale, réduire grandement le poids de la vidéo.

Capturer en standard ou en HD ? Pour réaliser des montages vidéo simples, sans détournage ni incrustation, une capture standard suffit (PAL 4/3 758 × 576). Mais si vous souhaitez réaliser des trucages avancés, tels que le détournage d'un sujet capturé sur fond vert, préférez une image haute définition (1 080 lignes), dont le nombre de points et la qualité de compression aideront à affiner le détournage, avant de réduire l'image ensuite à l'exportation vers Flash.

Quel que soit le format de capture, songez que la vidéo doit être contenue, au final, dans une fenêtre de navigateur de surface utile de 980 pixels par 550 pixels.

La lecture est-elle instantanée ? Il faut compter une mise en cache de quelques secondes avant que la vidéo ne puisse être jouée lorsqu'elle est diffusée sur le Web. Prévoyez cela dans la scénarisation de votre document Flash. Cette propriété peut toutefois être contrôlée par ActionScript

Intégrer ou non les vidéos dans le Flash ? La vidéo pour le Web est de préférence externalisée par rapport au document Flash qui s'y réfère. C'est le Flash qui fait référence au fichier vidéo *via* une instruction ActionScript. Cela permet d'alléger considérablement le poids du document Flash et en simplifie la maintenance. En outre, la vidéo peut être lue à

mesure de son chargement, ce qui n'est pas le cas d'un document qui intègre physiquement la vidéo.

Lorsque la vidéo est incluse physiquement dans le Flash, il faut en effet attendre le chargement complet du Flash avant de pouvoir lire le début de la vidéo. À quelques exceptions près (que nous abordons au Chapitre 8, section "Lire une vidéo en arrière"), la vidéo est toujours externalisée.

Une vidéo chargée dans le scénario implique également la recompilation du film à chaque publication, ce qui prend en outre beaucoup de temps.

Le format vidéo de Flash peut-il être lu sans Flash ? Le Flash gère les formats vidéo FLV (version 6 et ultérieur) et F4V (version 10 et ultérieur), voire certains formats QuickTime. Mais les vidéos Flash ne sauraient être lues indépendamment du document Flash qui les accompagne, car c'est lui qui contient le lecteur en mesure de gérer ce format. Quelques applications toutefois savent lire les formats vidéo Flash, comme le lecteur VLC, Real, Bridge et certaines versions de Quick Time. Le format F4V, que nous détaillons au Chapitre 7, est plus permissif cela dit que le FLV, du fait de son type de compression, basé sur le H-264.

Source compressée ou non compressée ? Préférez toujours exporter au format vidéo de Flash à partir d'une source non compressée. Les vidéos déjà compressées utilisent des algorithmes qui peuvent générer des parasites dans la vidéo de sortie. Une vidéo non compressée demeurera toujours de meilleure qualité au premier encodage.

Quelle configuration pour lire une vidéo sur le Web ? Pour lire une vidéo gérée *via* Flash, sur le Web, l'utilisateur requiert au moins l'ADSL ou une connexion Câble, un lecteur Flash de version 6 pour les vidéos FLV basiques, de version 8 pour les vidéos FLV composites avec alpha, et de version théoriquement supérieure à 10 pour la vidéo haute définition avec le format F4V. Dans tous les cas, une bonne carte vidéo est souhaitable, supérieure ou égale à 64 Mo. En dessous de 64 Mo, les flux vidéos paraîtront parfois saccadés pour des tailles même raisonnables. En dessous de 32 Mo, elles paraîtront saccadées quelle que soit leurs dimensions, même en réduisant la cadence à 12 ou 15 images par secondes.

Quelle différence entre un flux continu et un chargement progressif ? L'affichage d'une vidéo à chargement continu est traité à partir d'un serveur de streaming (littéralement, d'envoi de flux) qui fait varier la quantité et la qualité des données en fonction de la bande passante disponible, dynamiquement. Les paquets de données vidéos étant traités dynamiquement, ce système permet de modifier la nature du contenu selon la configuration et les interactions de l'utilisateur. Cette technique permet ainsi la diffusion d'émissions en direct et l'envoi d'un signal plus ou moins compressé selon la configuration matérielle du poste utilisateur. Le traitement d'un signal continu *via* Flash implique une solution serveur telle que Flash Media Server.

Une vidéo à chargement progressif est un fichier déjà enregistré au format FLV ou F4V. Le lecteur Flash la copie dans le cache du navigateur et lit la vidéo, à mesure de son chargement. Le traitement d'un flux en chargement progressif ne permet pas le direct, mais reste très simple à déployer, car aucun serveur spécifique n'est requis pour lire la vidéo. C'est cette dernière technique que nous utilisons dans cet ouvrage.

Quelle différence entre un fichier FLV et un fichier F4V ? Le FLV propose une compression forte au détriment de la qualité, mais il autorise une implémentation souple à l'intérieur d'une mise en forme graphique composée de multiples calques. Cela en favorise l'interactivité. Le FLV encodé en On2VP6 autorise par ailleurs la transparence. Étant disponible depuis Flash 6, il apporte également une grande compatibilité.

Le format F4V utilise une compression HD (codec H-264). Il est utilisé pour l'affichage de présentations vidéo linéaires et qualitatives (bandes-annonces, teasers, produits de luxe, animations 3D avec de la radiosit  ). Une vid  o F4V est th  oriquement seulement compatible avec Flash 10 et ult  rieur, mais nous d  montrons qu'elle peut   tre lue depuis une version plus ancienne au chapitre suivant. Ce format est privil  gi   pour les pr  sentations haut de gamme et pour une cible bien identifi  e qui sera   quip  e pour acc  der    ce type de contenu (qui dispose d'un lecteur Flash r  cent et d'une bonne carte vid  o).

Composition simple avec Apple Motion

Apple Motion est un logiciel de trucage vid  o et d'effets sp  ciaux inclus dans la suite Final Cut Studio d'Apple. Cet outil permet de r  aliser, entre autres, des habillages graphiques et anim  s    partir de formes vectorielles, d'images bitmaps, de particules g  n  r  es dynamiquement et de s  quences vid  o. Il est utilis   en t  l  vision pour les habillages d'  missions, pour l'interfa  ge de DVD et depuis peu pour le Web o   il apporte une vraie touche graphique gr  ce notamment    son puissant moteur de g  n  ration de particules et d'acc  l  ration.

Dans cette section, nous allons voir comment exporter, depuis Motion, une animation de particules d  j pr  d  finie, dans un format compatible avec l'encodage pour Flash (voir Figure 6.1). Dans cet exemple, l'animation de particules illustre l'  clat d'un feu d'artifice. Nous l'encoderons plus tard, avec sa transparence, pour Flash, et la placerons au-dessus d'une interface.

Figure 6.1

Aper  u de la vid  o Apple Motion apr  s publication.





Exemples > videoMotion > particules.motr

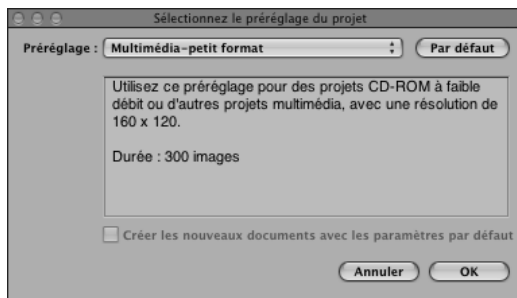
L'animation de particules que nous présentons ici est intégrée au logiciel Apple Motion. Pour les lecteurs qui ne disposeraient pas de l'application, une version exportée au format Quick Time est accessible dans le répertoire "videoMotion" des exercices du livre et se nomme "particules.mov". Vous pouvez directement passer à la section sur l'encodage si vous souhaitez intégrer des contenus vidéos FLV dans Flash.

Pour créer une animation vidéo de particules avec de la transparence, pour Flash et avec Motion, procédez comme suit :

1. Lancez l'application Apple Motion.
2. À l'ouverture, une boîte de dialogue de création de nouveau projet apparaît (voir Figure 6.2). Dans le menu Préréglages, sélectionnez l'option Personnaliser...

Figure 6.2

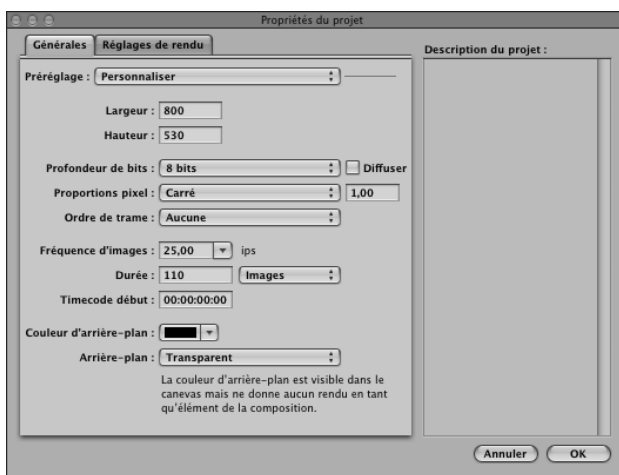
Fenêtre de nouveau projet dans Motion.



3. La fenêtre de Propriétés du projet apparaît (voir Figure 6.3).

Figure 6.3

Fenêtre de Propriétés du projet dans Motion.



4. Configurez un projet de résolution de 800 pixels de haut sur 530 pixels de large (ce qui correspond à notre surface de travail dans le document Flash), codé sur 8 bits (à savoir

la profondeur disponible sur les écrans informatiques tout public), de proportions de pixels Carré (soit un ratio de 1,00), sans trame. L'exemple que nous utilisons dure environ 110 images (durée de l'animation prédéfinie) et nous le cadencions à 25 images par secondes (25 ips, cadence standard pour la vidéo PAL en Europe). Nous sélectionnons enfin un arrière-plan de type Transparent, pour permettre plus tard la superposition de la vidéo avec d'autres éléments graphiques, au sein de l'application Flash.

Dans le nouveau projet, à gauche, figure la fenêtre Navigateur (voir Figure 6.4). Au centre, nous pouvons voir l'espace de travail, la scène. En haut, différents menus donnent accès à des effets qui peuvent être appliqués au contenu.

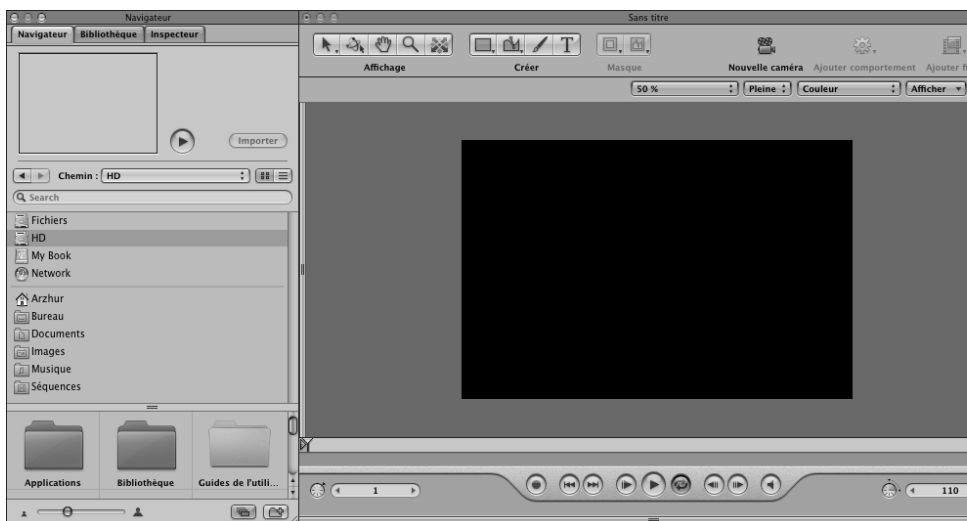


Figure 6.4

Fenêtre Navigateur.

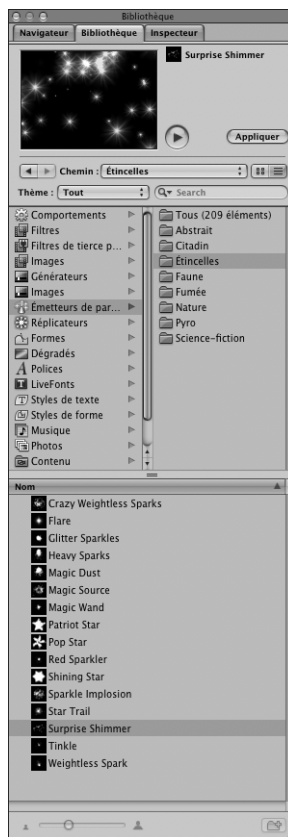
Dans la fenêtre Navigateur, vous pouvez identifier trois onglets : Navigateur, Bibliothèque et Inspecteur. Le Navigateur donne accès aux fichiers et dossiers de votre système. La Bibliothèque met à disposition un certain nombre d'objets prédéfinis et prêts à l'emploi. L'Inspecteur, lui, affiche toutes les options de contrôle disponibles pour l'objet activé dans la scène (comme l'Inspecteur de propriétés de Flash).

Cliquez sur l'onglet Bibliothèque. Puis, dans la partie inférieure, sélectionnez le répertoire Émetteur de particules. Dans la colonne de droite, sélectionnez maintenant le dossier Étincelles. Puis, dans la partie inférieure de la fenêtre, cliquez sur l'animation prédéfinie nommée "Surprise Shimmer" (voir Figure 6.5).

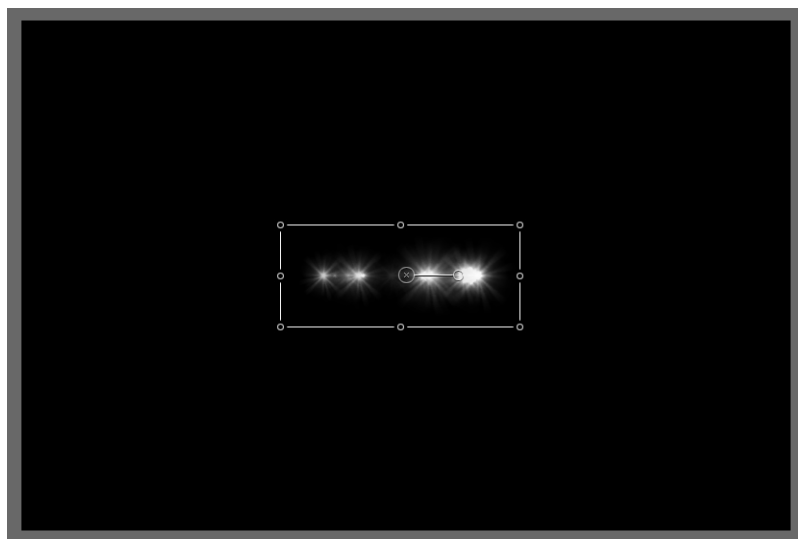
Un aperçu de l'animation est disponible au sommet de la fenêtre Navigateur. En bas de l'écran, vérifiez que la tête de lecture du scénario est bien située au début de l'animation (voir Figure 6.7).

Figure 6.5

Sélection d'une animation prédéfinie.

**Figure 6.7**

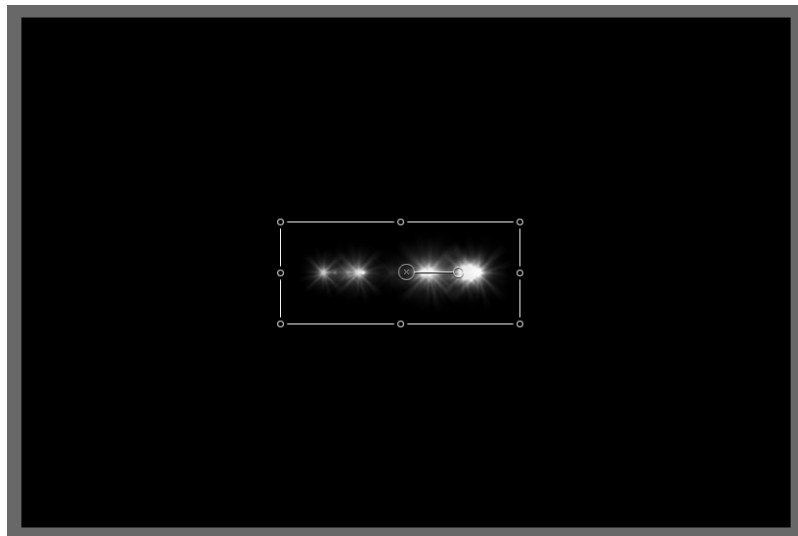
Fenêtre de scénario.



Glisser-déposez l'animation sélectionnée de la fenêtre Navigateur vers le centre de la scène (voir Figure 6.7). Puis, appuyez sur la barre d'espace pour tester l'animation.

Figure 6.7

Animation
déposée.



Un scintillement d'étoiles s'anime dans la scène courante. L'animation fonctionne. Vous pouvez enregistrer le document de travail et l'exporter vers un format compatible pour l'encodage Flash.

Faites Fichier > Enregistrer. Puis, confirmez l'enregistrement du document Motion au format natif (.mtn) dans le dossier de votre choix. Par exemple, dans le répertoire Sources ou dans le dossier *videoMotion* des exercices du livre. Ce fichier natif ne sera pas mis en ligne. Il servira uniquement à modifier éventuellement votre création avant de l'exporter à nouveau dans un format aplati et compatible.

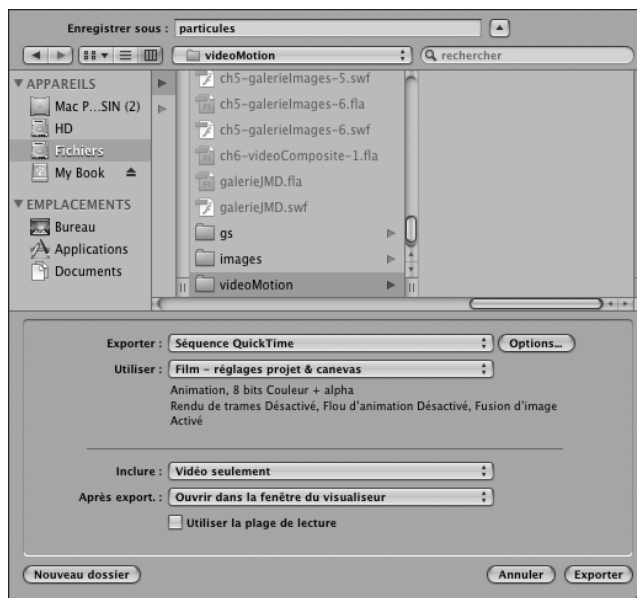
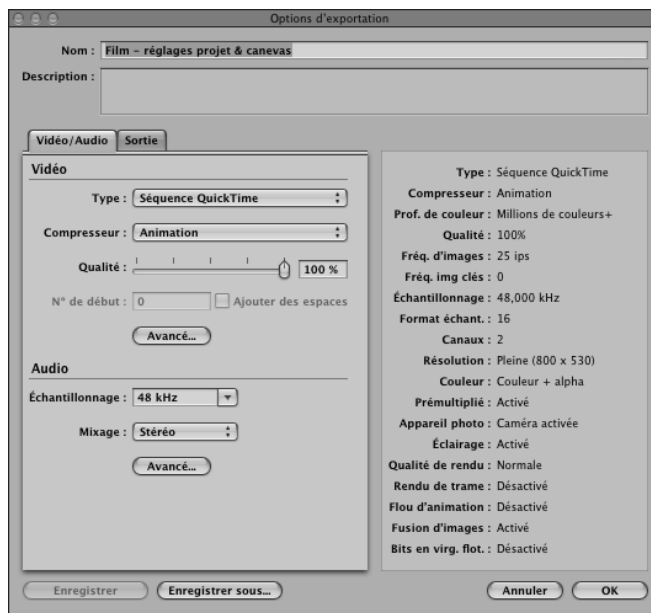
Vous pouvez maintenant exporter la vidéo pour le Web.

Dans Motion, il n'y a pas d'exportation directe au format Flash FLV ou F4V. Pour encoder la vidéo au format Flash, nous utilisons l'application Adobe Media Encoder. Mais, pour ce faire, nous devons d'abord exporter la vidéo dans un format standard, aplati et qui préserve la transparence. Nous choisissons de l'exporter au format usuel Animation de Quick Time :

1. Faites Fichier > Exporter. Puis, dans la boîte de dialogue, nommez la vidéo Particules. Plus bas, dans le menu Exporter, choisissez Quick Time. À droite, cliquez sur le bouton Options (voir Figure 6.8).
2. Dans les options d'exportation, choisissez une compression de type Animation en préservant au maximum la qualité de l'animation (voir Figure 6.9). Vous remarquez, dans le détail des paramètres de compression à droite, que la couche alpha est automatiquement intégrée dans ce format. Validez toutes les fenêtres. Puis, quittez l'application.

Figure 6.8

Fenêtre Exporter.

**Figure 6.9**Options
de compression.

La vidéo est maintenant exportée au format Quick Time et peut être encodée pour Flash avec Adobe Media Encoder. Reportez-vous, plus loin dans ce chapitre, à la section consacrée à l'enregistrement avec l'encodeur Adobe pour publier la vidéo au format Flash FLV.

Un autre exemple est disponible dans le dossier des exemples du livre et se nomme "galaxie.mov". Le fichier Motion associé se nomme "galaxie.mtn".

Encoder en FLV avec Apple Compressor

Compressor est livré en standard dans la suite Final Cut Studio de Apple. Pour les utilisateurs de Compressor, vous pouvez encoder au format Flash vidéo directement depuis le logiciel Compressor. Vous devez, depuis Compressor, exporter la vidéo au format Quick Time, et dans les options d'encodage du format Quick Time, sélectionner le format FLV.

À retenir

- Apple Motion permet d'exporter au format Flash vidéo FLV avec de la transparence, à condition de le publier d'abord au format Quick Time Animation, puis de l'encoder ensuite avec l'encodeur Adobe au format FLV.
- Apple Motion complète Adobe After Effects en cela qu'il intègre un puissant moteur d'interpolation des images et un bon générateur de particules.
- Les vidéos Apple peuvent également être exportées *via* Compressor pour Flash, avec le même paramètre d'exportation Quick Time pour l'animation, mais l'option FLV est ici disponible.

Composition simple avec Adobe After Effects

Comme nous l'avons vu avec Motion, nous allons voir comment exporter une animation simple réalisée ici avec After Effects, pour Flash.

After Effects est un logiciel de composition et d'effets spéciaux, disponible dans la suite vidéo Adobe. Ce logiciel, similaire à Motion, permet de réaliser des animations spatiales en 3D de manière plus avancée que Motion puisqu'il peut intégrer des objets modélisés en 3D, mais il ne dispose pas d'un moteur de particules aussi gratifiant. Les deux applications se complètent donc assez bien dans la production d'effets visuels. Il est à ce titre possible d'importer une vidéo réalisée par Motion dans After Effects et inversement, pour l'enrichir des fonctionnalités de l'autre application.

Dans cette section, nous allons voir comment créer une animation et la publier directement au format FLV ou F4V pour Flash, depuis After Effects.



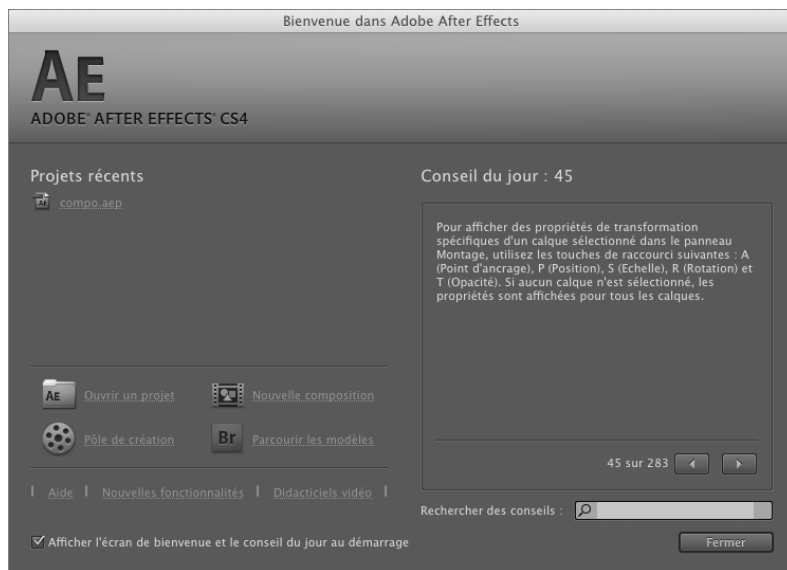
Exemples > videoAfterEffects > flocons.aep

Pour les lecteurs qui ne disposeraient pas de l'application After Effects, une version exportée de la vidéo est disponible dans le répertoire videoAfterEffects et se nomme flocons.mov. L'animation que nous allons étudier représente une pluie de flocons de neige déjà mise en forme. Mais avant d'ouvrir ce document, nous allons voir comment configurer un projet After Effects pour Flash :

1. Pour créer un document After Effects pour le Web, lancez l'application After Effects.
2. Au démarrage de l'application, une fenêtre d'accueil apparaît (voir Figure 6.10).

Figure 6.10

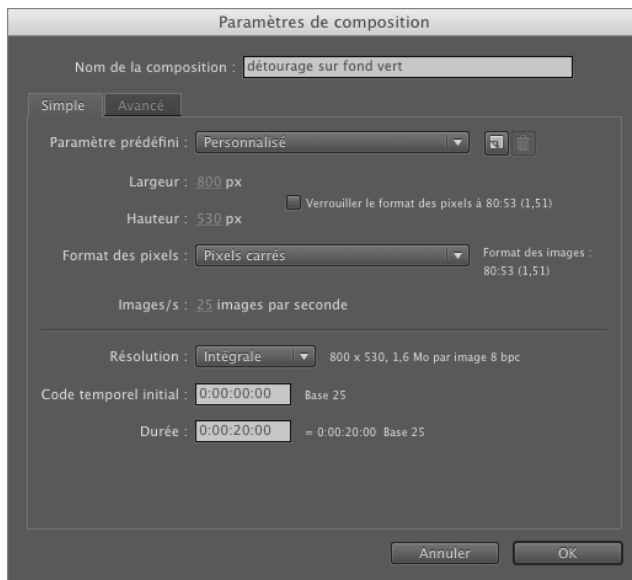
Fenêtre d'accueil
d'After Effects.



3. Cliquez sur Nouvelle composition pour créer une nouvelle séquence de vidéo composite.
4. Une boîte de dialogue apparaît (voir Figure 6.11).

Figure 6.11

Paramètres de
composition.



5. Dans la boîte de dialogue des paramètres de la nouvelle composition, spécifiez une configuration personnalisée de largeur 800 pixels et de hauteur 530 pixels (dimensions disponibles dans notre document Flash). Puis, choisissez un format de pixels carrés et une cadence à 25 ips, comme vu précédemment avec Motion. Spécifiez enfin une durée

souhaitée en secondes. Par exemple, pour une séquence d'une durée de 20 secondes, inscrivez 0:00:20:00 dans le champ correspondant. Validez.

After Effects affiche une nouvelle composition (voir Figure 6.12) basée sur des pixels carrés. Vous pouvez réaliser vos effets, importer des médias, multiplier les calques et animer les propriétés.

Figure 6.12

Nouvelle composition.



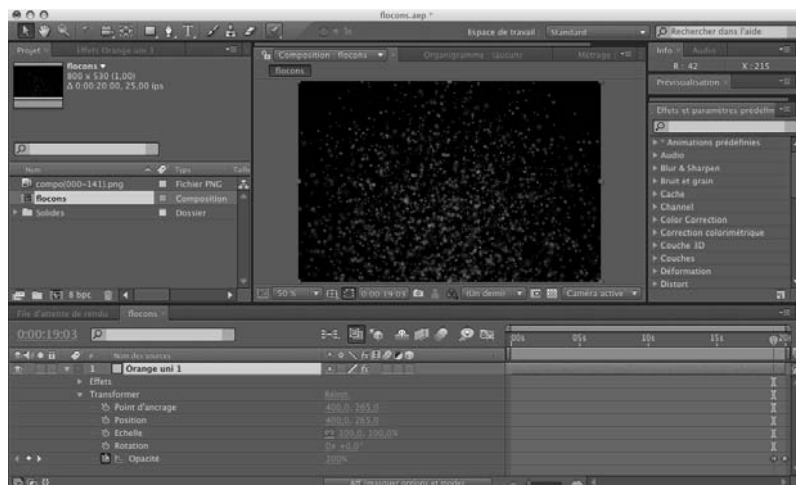
Dans le dossier des exemples du livre, une animation de flocons est déjà réalisée. Nous allons l'ouvrir pour l'exporter directement au format vidéo de Flash. Ouvrez le projet After Effects intitulé "flocons.aep", enregistré au format natif dans le dossier "videoAfterEffects".

Dans ce document (voir Figure 6.13), un solide est placé sur la scène et utilise un effet de particules (Effets > Simulation > CC Particles Systems II). L'effet neutralise le solide, mais l'utilise pour exister en tant qu'objet dans le scénario. Les options des effets sont accessibles dans le scénario et dans la fenêtre Effets (Fenêtre > Effets) et l'animation de fondu en sortie (fondu au noir) est matérialisée dans le scénario avec des images-clés appliquées à la propriété Opacité.

Pour exporter la composition directement au format vidéo de Flash, activez d'abord la séquence à exporter en cliquant sur la scène (où l'on aperçoit la vidéo). Faites Fichier > Exporter > Flash Vidéo FLV... Un message vous avertit que vous disposerez de plus d'options de contrôle en exportant depuis la fenêtre de rendu de After Effects. L'export *via* le menu Fichier utilise en effet une ancienne version de l'encodeur Flash vidéo, mais tout à fait conforme aux besoins d'une vidéo composite de base, avec alpha, et compatible Flash 8. Nous reviendrons sur les autres options dans la section "Échantillonner la vidéo pour Flash".

Figure 6.13

Composition
flocons.aep.



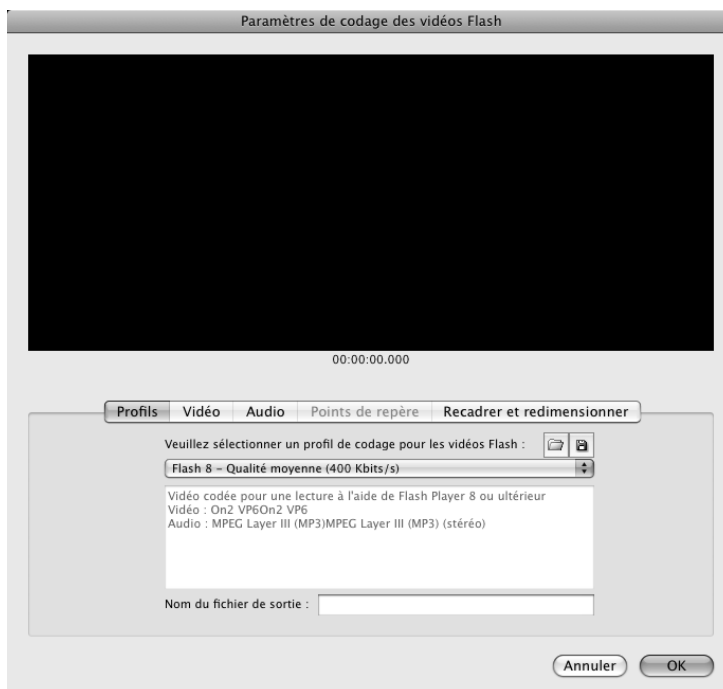
Validez pour confirmer l'enregistrement.

Dans la boîte de dialogue, vous pouvez accéder à différentes options d'échantillonnage (voir Figure 6.14), mais limitées.

Dans la liste des compressions prédéfinies située dans la partie inférieure de la fenêtre, vous accédez à différents formats d'encodage plus ou moins compatibles avec d'anciennes versions du lecteur Flash. Seuls les formats compatibles Flash 8 autorisent l'encodage de la couche alpha qui préserve la transparence de la vidéo dans Flash.

Figure 6.14

Paramètres
d'encodage des
vidéos Flash.



Pour un encodage qui autorise la transparence, sélectionnez Flash 8, dans la liste déroulante. Puis, adaptez votre choix selon l’option qui correspond le mieux à la bande passante définie pour votre cible. Reportez-vous au Tableau 6.1 pour connaître les valeurs à sélectionner en fonction de votre cible.

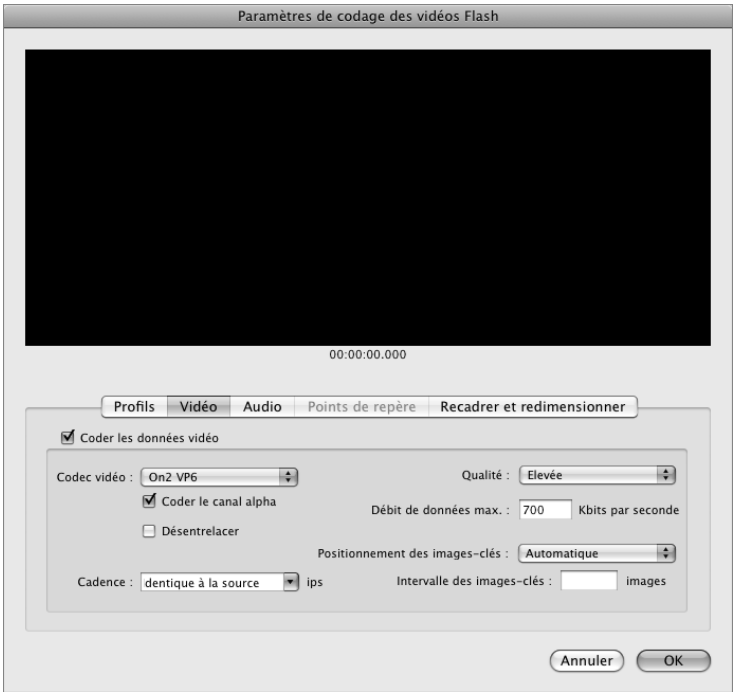
Tableau 6.1 : Compression et débit.

Débit	Compression
Utilisateurs Modem 56k	Pas de signal vidéo fluide.
Utilisateurs ADSL 1	400 kbps.
Utilisateurs ADSL 2 (majorité des utilisateurs)	700 kbps.
Utilisateurs Super ADSL, Câble, fibre optique	2 048 kbps à 4 096 kbps.
Utilisateurs locaux (non connectés, lecture en locale sur le poste utilisateur, CD, DVD)	> 4 096 kbps.

Dans la catégorie Vidéo (voir Figure 6.15), cochez l’option Coder le canal Alpha, puis validez.

Figure 6.15

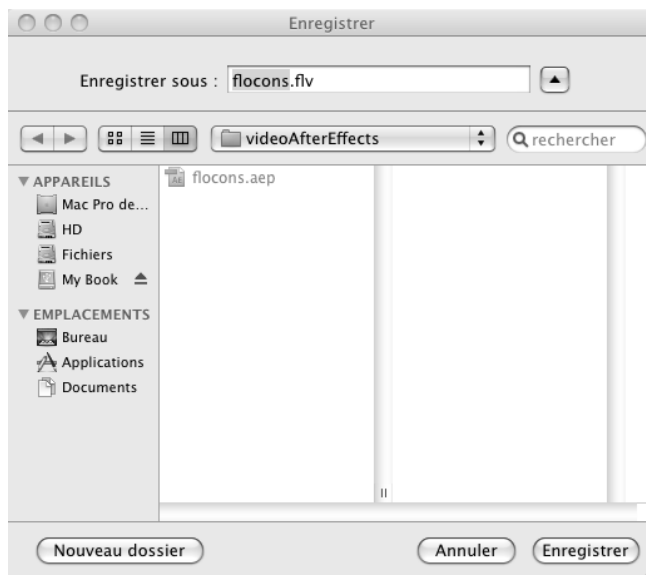
Coder le canal Alpha.



La fenêtre d’enregistrement apparaît. Nommez la vidéo flocons.flv, par exemple, puis validez (voir Figure 6.16).

Figure 6.16

Enregistrer la vidéo au format FLV.



La vidéo est enregistrée au format FLV. Vous pouvez l'intégrer dans Flash.

Pour le descriptif détaillé des options de compression du format FLV, reportez-vous à la section suivante. Pour savoir comment enregistrer pour Flash depuis d'autres logiciels vidéo (Final Cut Pro, Premiere Pro, mais aussi iMovie ou Window Movie Maker), reportez-vous au Chapitre 7.

Trucage avec captation sur fond vert

Si vous savez exporter une vidéo avec la couche alpha, vous savez aussi intégrer un sujet capturé sur fond vert. Pour réaliser une incrustation avec fond vert, relevez ce qui suit :

- Lors de la captation, veillez à obtenir un fond dont la couleur n'est pas spécifiquement verte, mais dont la teinte s'oppose en toutes circonstances à celle du sujet capturé.
- Utilisez un fond sans aspérité et régulier, qui réfléchisse bien la lumière en direction de la caméra et sans créer d'ombre ni laisser apparaître de plis.
- Rendez le fond homogène en l'éclairant sous plusieurs angles avec des lumières diffuses. Il ne doit pas y avoir de halo ou de zone moins exposée qu'une autre. Attention, une lumière trop forte génère un halo jaune (image brûlée).
- Éclairez le sujet pour compenser le contre-jour que vous obtenez avec un fond exposé. Préférez un éclairage indirect du sujet, pour éviter l'apparition au sol ou sur le fond vert, d'ombres portées.
- Effectuez ensuite, si possible, un enregistrement en haute définition, avec de bonnes optiques, en image progressive et avec la compression la moins forte possible (donc, surtout pas en DV ni en HDV qui sont des formats très compressés), de sorte à restituer un piqué d'image le plus fin possible. Cela facilitera le détournage dans le logiciel de compositing.

- Pour détourer un fond vert, dans After Effects, vous pouvez utiliser les effets Keying > Keylight ou les effets du menu Effets > Correction colorimétrique, et les dégrossir par exemple avec un masque (pour supprimer d'abord les bords du cadre, les câblages qui entrent dans le champ, etc.). Dans Motion, dégrossissez de même avec un masque et utilisez Ajouter un filtre > Incrustation > Écran bleu ou vert pour le détourage.
- Réduisez éventuellement l'échelle de l'image, après le détourage bien sûr, juste avant de l'encoder au format vidéo de Flash.
- Encodez au format FLV avec l'option Coder le canal alpha.

Exporter le projet After Effects natif directement vers Flash

Si la composition After Effects doit utiliser des objets que vous souhaitez rendre interactifs dans Flash, vous pouvez exporter ces contenus directement vers Flash, sans rendre ni aplatir la vidéo. Faites Fichier > Exporter > Exporter au format Flash Professionnel XFL. Puis, depuis Flash, ouvrez le document XFL et enregistrez la composition au format FLA. Attention, tous les effets ne sont pas toujours bien interprétés. De plus, si des calculs doivent être appliqués à des animations de vidéos, un rendu sera malgré tout nécessaire avant de pouvoir importer le projet dans Flash. Le document Flash obtenu intègre physiquement la vidéo, ce qui en limite naturellement l'exploitation puisque nous savons qu'elle est mieux gérée en externe. Pour de plus amples informations néanmoins sur ces passerelles, consultez le livre *D'After Effects à Flash / De Flash à After Effects*, de Richard Harrington et Marcus Geduld, publié aux éditions Pearson, particulièrement adapté aux utilisateurs de Flash qui souhaitent par ailleurs apprivoiser After Effects.

À retenir

- Adobe After Effects offre une bonne transversalité avec Flash et permet d'exporter facilement au format FLV avec de la transparence.
- Il est possible d'exporter une composition After Effects directement vers Flash sans calcul de rendu, si les objets animés l'autorisent.

Échantillonner la vidéo pour Flash

Dans cette section, nous abordons l'encodage de fichiers vidéo, à partir d'une source déjà enregistrée (Quick Time, AVI, DV, etc.) ou depuis une application standard de création de contenu vidéo. Parmi celles-ci, nous abordons les solutions professionnelles des suites Adobe et Apple avec After Effects, Premiere Pro et Final Cut Pro, ainsi que les solutions embarquées sur les systèmes Mac et PC avec iMovie et Window Movie Maker.

À l'issue de ce chapitre, vous serez en mesure de convertir tout type de montage vidéo en format de fichier vidéo prêt à être intégré à Flash.

Encoder en FLV avec After Effects

After Effects possède la même fenêtre de compression que celle disponible dans l'encodeur Adobe :

1. Pour l'ouvrir, faites Composition > Compiler le film.
2. Lors du premier enregistrement, After Effects demande d'enregistrer au format Quick Time. Dans ce cas, confirmez. Rien ne se produit. Puis revenez dans la file de rendu en faisant, de nouveau et si nécessaire, Composition > Compiler le film.
3. La file d'attente de rendu apparaît.
4. Dans la file de rendu, cliquez sur le lien jaune intitulé Module de sortie non destructif.
5. Dans le menu Format de la nouvelle boîte de dialogue, choisissez FLV.
6. Puis, dans Sortie vidéo, cliquez sur Options de format.

Reportez-vous ensuite à la section "Encoder avec Adobe Media Encoder" pour le détail des réglages.

Encoder en FLV avec Premiere Pro

Dans Premiere Pro, il est possible d'exporter plus directement au format vidéo de Flash :

1. Faites Fichier > Exportation > Médias.
2. Puis, en haut et à droite, dans la fenêtre de dialogue, dans le menu Format, choisissez FLV|F4V.
3. Dans l'onglet Multiplexeur enfin, cochez l'option FLV.

Reportez-vous ensuite à la section "Encoder avec Adobe Media Encoder" pour le détail des réglages.

Encoder en FLV avec Final Cut Pro

Avec Final Cut Pro, vous devez d'abord enregistrer la vidéo dans un format standard avant de l'encoder avec Adobe Media Encoder ou Compressor.

1. Faites directement Fichier > Exporter > Exporter *via* la conversion Quick Time.
2. Puis, dans les options de réglage vidéo de Quick Time, choisissez Animation.
3. Confirmez l'enregistrement au format Quick Time.

Reportez-vous ensuite à la section "Encoder avec Adobe Media Encoder" pour convertir un fichier Quick Time en FLV.

Encoder en AVI-DV avec Window Movie Maker

Sur Windows, avec Window Movie Maker, vous pouvez publier au format Window Media ou AVI. Nous utilisons le format AVI avec le codec DV pour une compatibilité transversale avec la suite Adobe.

1. Une fois votre montage réalisé, faites Fichier > Enregistrer le fichier vidéo.
2. Puis, choisissez de l'enregistrer sur votre poste de travail.
3. Confirmez son nom et l'emplacement de l'enregistrement.
4. Puis, dans les options de configuration, sélectionnez l'option Autres paramètres et choisissez le format DV-AVI (PAL). Validez l'encodage.

Le fichier vidéo obtenu peut être maintenant encodé avec Adobe Media Encoder.

Encoder en MOV ou DV avec iMovie

Dans iMovie, vous pouvez accéder directement aux éléments sources des séquences capturées, avant montage, dans leur format natif. Ils sont disponibles directement dans le système (Finder).

1. Sur chaque plan disponible dans iMovie, faites un clic-droit (ou Ctrl+Clic) puis, sélectionnez l'option Afficher dans le Finder.
2. Vous pouvez aussi exporter le montage dans un format compatible avec l'encodeur vidéo Adobe. Faites Exporter > Exporter à l'aide de Quick Time.
3. Puis, dans les options de réglage vidéo de Quick Time, sélectionnez le format Animation.

Reportez-vous ensuite à la section "Encoder avec Adobe Media Encoder" pour convertir un fichier Quick Time en FLV.

Encoder en FLV avec Adobe Media Encoder

Quel que soit le logiciel de montage ou de trucage vidéo employé pour créer le fichier vidéo, et même si ce logiciel ne vous offre pas la possibilité d'exporter la vidéo directement au format Flash, vous pouvez l'encoder avec Adobe Media Encoder. Cet utilitaire d'encodage de médias est livré dans la suite Adobe et disponible dans vos programmes, même si vous n'avez installé que Flash.

L'encodeur offre, en plus des formats vidéo requis pour Flash, tous les formats vidéo et audio que d'autres applications éventuellement installées sur votre machine peuvent proposer. Ainsi, si vous avez installé toute la suite vidéo de Adobe, vous aurez accès, à travers l'encodeur, aux codecs distribués par ces autres applications. Il en va de même avec l'ensemble des codecs distribués par la suite Apple.

Les réglages dont nous disposons à travers l'encoder Adobe sont identiques à ceux disponibles depuis les autres logiciels de la suite. En étudiant les réglages dans cette section, vous serez donc en mesure d'utiliser aussi les options d'exportation avancées des logiciels Premiere Pro et After Effects pour les formats vidéo de Flash FLV et déjà une partie de l'encodage pour le format F4V.

Les formats pris en charge par Adobe Media Encoder

Les formats pris en charge par l'encodeur sont désignés par leurs codec de compression et non par leur extension. Dans un format vidéo, nous distinguons la coquille de la vidéo (Quick Time, AVI) et le format d'encodage du fichier contenu dans l'enveloppe vidéo elle-même (Animation, H-264, DV). C'est la raison pour laquelle Flash peut parfois interpréter des fichiers Quick Time sans encodage spécifique supplémentaire, si l'encodage utilisé est identique à celui d'un fichier FLV ou F4V.

Si l'encodeur Adobe gère différents formats, il n'autorise pas toutes les options d'encodage pour l'ensemble de ces formats. Le format le plus ouvert reste le format QuickTime (MOV) avec une compression de type Séquence animée (Animation).

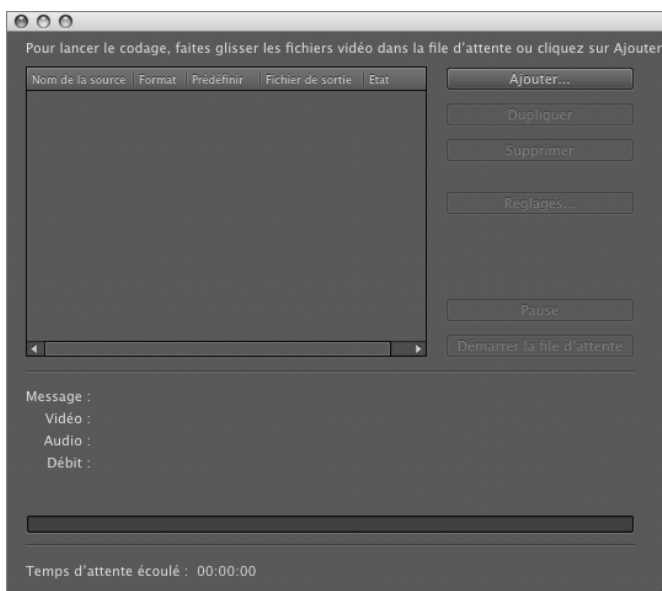
Voici la liste des formats pris en charge par l'encodeur Adobe :

- **Vidéo.** 3G2, GIF animé (GIF), DLX (Sony, Windows uniquement), DV (dans un conteneur MOV ou AVI), FLV, F4V, M2T (Sony HDV), QuickTime (MOV), MP4 (XDCAM EX), Formats MPEG-1, MPEG-2 et MPEG-4 (MPEG, MPE, MPG, M2V, MPA, MP2, M2A, MPV, M2P, M2T, AC3, MP4, M4V, M4A). Certains formats de données MPEG sont enregistrés dans des conteneurs dont le format n'est pas reconnu par Adobe Media Encoder : les extensions *.vob* et *.mod* sont notamment concernées. Dans certains cas, vous pouvez modifier l'extension des fichiers afin de les importer dans Adobe Media Encoder sous un format reconnu. MTS (AVCHD), Media eXchange Format (MXF). Uniquement certains types (une variante Op-Atom utilisée par les caméscopes Panasonic DV, DVCPRO, DVCPRO50 et DVCPRO HD pour les enregistrements sur support Panasonic P2). Adobe Media Encoder peut également importer des fichiers XDCAM HD au format MXF. Netshow (ASF, Windows uniquement), Vidéo pour Windows (AVI, WAV ; requiert QuickTime sous Mac OS), ne peut pas importer de fichiers vidéo DivX®, ni de fichiers AVI codés avec DivX. WMV (WMV, WMA, ASF ; Windows uniquement).
- **Audio.** Fichier Adobe Sound (ASND), AAC (Advanced Audio Coding, M4A), AIF, AIFF (*Audio Interchange File Format*), AVI (*Audio Video Interleaved*), WAV (*Audio WAVEform*), MP3 (MP3, MPEG, MPG, MPA, MPE), MOV, Windows Media Audio (WMA, Windows uniquement), Vidéo pour Windows (AVI, WAV, requiert QuickTime sous Mac OS X).
- **Images fixes.** Adobe Media Encoder prend en charge les fichiers d'images fixes 8 bits par canal (4 octets par pixel) et 16 bits par canal (8 octets par pixel). Il convertit les images de résolution inférieure en 8 bits par canal et celles de résolution supérieure en 16 bits par canal lors de l'importation. Les fichiers à résolution élevée sont pris en charge à une seule virgule flottante en simple précision par canal (16 octets par pixel). Adobe Photoshop et séquence Photoshop (PSD), Bitmap et séquence Bitmap (BMP, DIB, RLE), GIF, Fichier icône (ICO) (Windows uniquement), JPEG et séquence JPEG (JPE, JPG, JFIF), PICT et PICT (PIC, PCT), Portable Network Graphics (PNG), Targa et séquence Targa (TGA, ICB, VDA, VST), TIFF et séquence TIFF (TIF). Vous pouvez importer des fichiers Illustrator et Photoshop à calques sous forme de séquences.
- **Montages aux formats natifs.** Adobe Premiere Pro (PRPROJ), Projet After Effects (AEP).

Pour encoder tout type de vidéo au format Flash FLV (ou F4V), lancez l'application Adobe Media Encoder (voir Figure 6.17).

Figure 6.17

Adobe Media
Encoder.

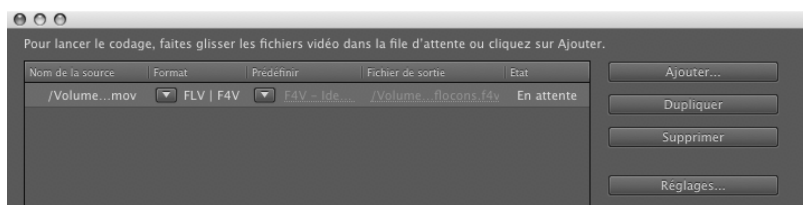


Cette application gère, dans la partie supérieure, une liste de documents à encoder. Dans la partie inférieure, nous pouvons voir la progression de l'encodage pour chaque fichier rendu individuellement. Puis, à droite, des options de réglage qui permettent de personnaliser la compression (dimensions, débit, transparence, etc.).

Pour encoder une vidéo, vous devez l'ajouter dans la liste de rendu. Pour cela, cliquez sur le bouton Ajouter, situé à droite ou bien glissez-déposez directement le fichier prêt à encoder dans la file de rendu, située dans la partie supérieure de l'application (voir Figure 6.18).

Figure 6.18

Ajouter une vidéo
à la file de rendu.



La fenêtre affiche le nom du document ajouté et propose quelques réglages prédéfinis de compression. Pour chaque vidéo ajoutée, différents paramètres sont disponibles : le nom de la source et son positionnement sur votre machine, le format, des réglages prédéfinis pour le format choisi, le nom, le chemin du fichier de sortie et l'état encodé – en cours d'encodage ou en attente d'encodage – qui lui est associé.

Dans la colonne Format, sélectionnez l'option FLV|F4V si cette option n'était pas déjà active par défaut.

Dans la colonne Prédéfinir, activez l'un des différents réglages prêts à utiliser.

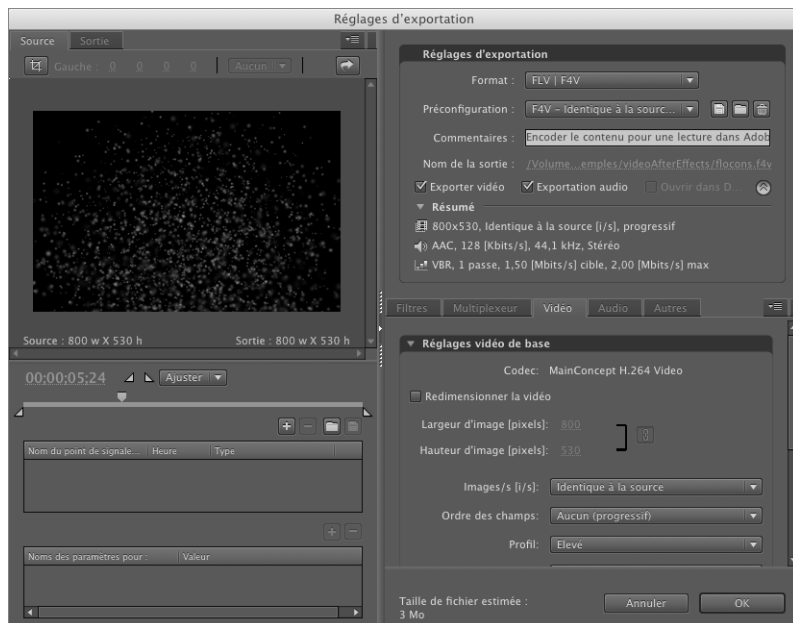
Réglages d'exportation

La fenêtre de rendu qui s'affiche au lancement de l'application permet d'accéder aux détails des réglages d'exportation pour affiner les réglages proposés par défaut.

Dans la file d'attente de rendu (voir Figure 6.18), dans la colonne Prédéfinir, cliquez sur le lien jaune pour accéder aux options de réglage du format sélectionné. Une nouvelle boîte de dialogue s'ouvre (voir Figure 6.19).

Figure 6.19

Réglages
d'exportation.



À l'intérieur de cette fenêtre, nous distinguons deux parties. À gauche, un aperçu permet de visualiser, recadrer et définir des points de repère qui permettent d'ajouter une couche d'interactivité dans une vidéo. À droite, les options d'échantillonnage gèrent la compression audio et vidéo. Nous abordons l'ensemble de ces réglages, par catégorie, dans les sections qui suivent.

Source et Sortie

L'onglet Source, situé en haut et à gauche de la fenêtre de réglages d'exportation, affiche d'abord un aperçu de la vidéo. Pour visualiser l'ensemble de la vidéo, vous pouvez déplacer la tête de lecture qui se trouve sous la zone d'affichage de la vidéo, le long de la bande jaune qui représente la durée de la vidéo.

Vous pouvez également modifier les points d'entrée et de sortie de la vidéo de sorte à rogner les premières et les dernières images de la séquence, pour en réduire la durée. Pour cela, déplacez les triangles situés à l'extrémité gauche et droite de la bande jaune et rapprochez-les vers le milieu de cette bande, jusqu'à l'endroit à partir duquel vous souhaitez démarrer et interrompre la vidéo.

Au-dessus de la bande jaune, un menu Ajuster permet de visualiser la vidéo en taille réelle, telle qu'elle apparaîtra dans Flash, ou bien avec d'autres proportions. Pour un aperçu réel, sélectionnez 100 %. Pour un aperçu intégral, sélectionnez Ajuster.

Dans la partie inférieure, vous avez la possibilité de créer des points de repère dans le flux vidéo. Nous reviendrons sur cette notion au Chapitre 8.

Au sommet enfin, un bouton de recadrage permet de supprimer les bords de la vidéo et n'en conserver qu'une partie. Pour recadrer, cliquez sur ce bouton, puis dessinez un rectangle sur la zone d'affichage de la vidéo (voir Figure 6.20). Pour valider le recadrage, passez à l'onglet Sortie, puis sélectionnez l'option Modifier la taille de la sortie, du menu intitulé Réglage du recadrage (voir Figure 6.21).

Figure 6.20

Source.

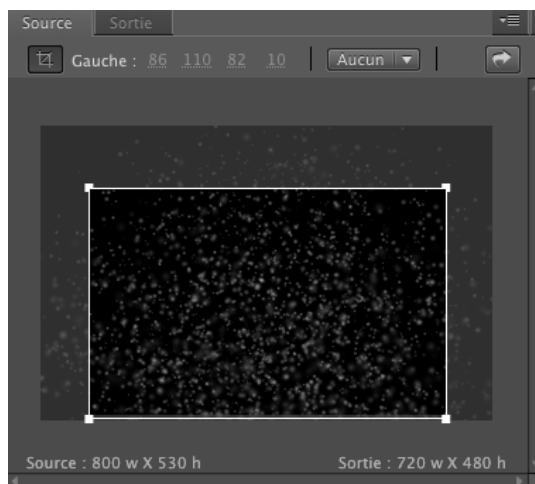
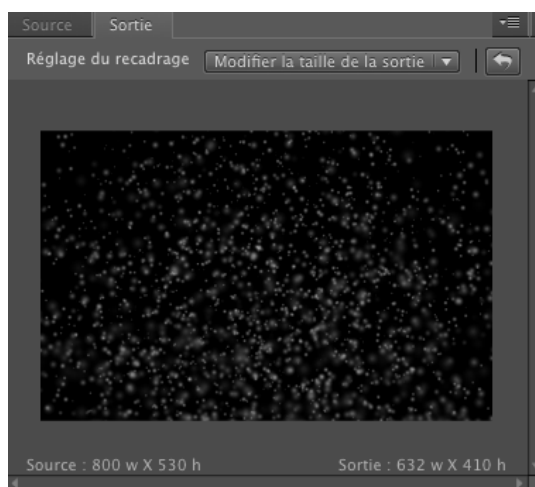


Figure 6.21

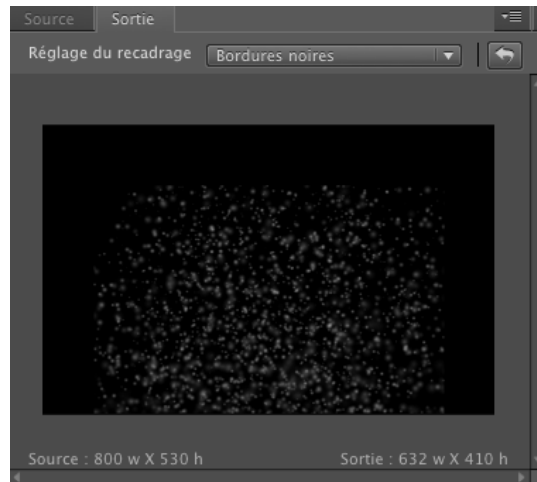
Sortie avec option
Modifier la taille
de la sortie.



Dans ce même menu, l'option Bandes noires conserve les dimensions initiales de la vidéo, mais remplit la zone supprimée par du noir (voir Figure 6.22).

Figure 6.22

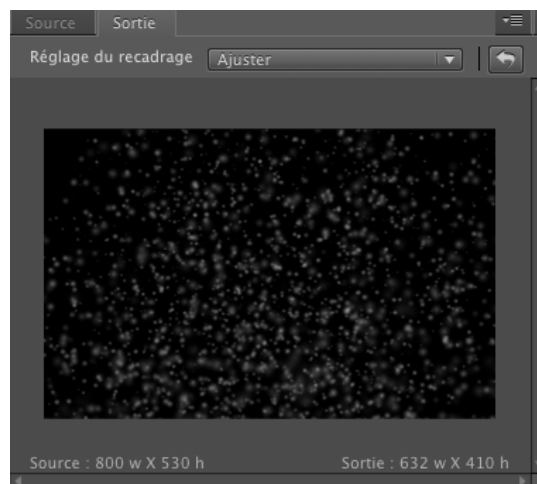
Sortie avec option
Bordures noires.



En sélectionnant l'option Ajuster, l'image recadrée épouse les dimensions initiales de la vidéo. Cette option agrandit la vidéo et par conséquent la détériore de manière importante (voir Figure 6.23).

Figure 6.23

Sortie avec option
Ajuster.



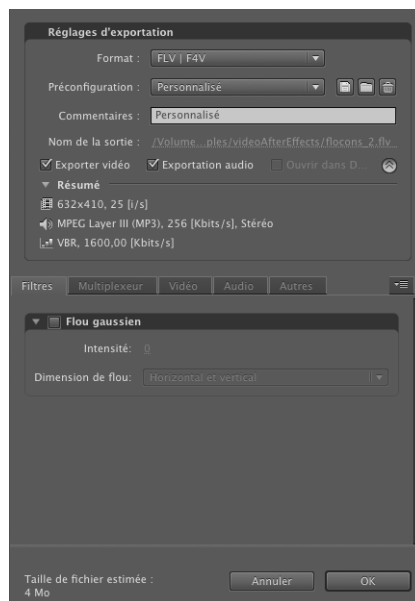
Une fois les premiers réglages de rognage définis, vous pouvez personnaliser les paramètres de l'encodage, affichés à droite de la fenêtre.

Réglages personnalisés

À droite de la fenêtre de réglages d'exportation, nous pouvons contrôler les paramètres d'échantillonnage pour la compression du flux vidéo en cours (voir Figure 6.24).

Figure 6.24

Réglages d'exportation personnalisés.



En haut de la fenêtre figurent les réglages de base. Dans la partie inférieure, vous pouvez modifier les paramètres attribués par défaut, des réglages prédéfinis :

- Dans le menu Format, vous pouvez revenir sur le format de fichier. Conservez l'option FLV|F4V pour le format vidéo de Flash.
- Dans le menu Préconfiguration, vous pouvez de nouveau accéder à des réglages préenregistrés, comme la fenêtre de rendu. Conservez le réglage actuel que nous allons modifier en intervenant sur les réglages en bas de la fenêtre.
- Le lien jaune, Nom de la sortie, permet de renommer le fichier qui sera créé lors de l'encodage. Puisque le document est un élément destiné au Web, nommez-le sans espace ni caractères spéciaux ou accentués et avec son extension .flv ou .f4v, selon le format d'encodage choisi. Conservez l'option Exporter vidéo, pour exporter le signal vidéo lors de l'encodage. Conservez l'option Audio, pour conserver le son à l'encodage.



Utiliser le FLV pour l'audio, la vidéo ou les deux. Le format FLV est initialement prévu pour gérer de la vidéo, et donc, indirectement, le son. Mais il est possible d'exploiter les propriétés d'un flux FLV pour encoder uniquement du son, uniquement de la vidéo ou les deux à la fois. Par exemple, vous pouvez traiter pour une émission audio seule sous la forme d'un flux vidéo, en désactivant l'option Exporter vidéo. Ce procédé est commode car il évite de gérer le son avec ActionScript et per-

met, en outre, de l'accompagner d'interactivité grâce à la synchronisation d'actions avec les points de repère disponibles uniquement avec le format vidéo de Flash (voir aussi le Chapitre 8).

Une fois ces préréglages vérifiés, vous pouvez les personnaliser dans la partie inférieure de la fenêtre où sont affichés cinq onglets : Filtres, Multiplexeur, Vidéo, Audio et Autres.

Filtres

L'onglet Filtres permet de flouter l'image. Cela peut adoucir, par exemple, une image initialement très dégradée. Mais cette option est surtout utilisée pour réduire le bruit d'une vidéo et accélérer le calcul du rendu. Le filtre flou altère, bien entendu, l'image finale.

Pour activer le flou, cochez l'option Flou Gaussien. Puis affinez les réglages situés au-dessous de l'option activée.

Multiplexeur

La catégorie Multiplexeur permet de choisir le type d'algorithme que l'encodeur appliquera à la vidéo destinée à Flash. Deux options sont proposées : FLV ou F4V. De ce choix dépend la compatibilité de la vidéo avec les versions antérieures du lecteur Flash. FLV permet de gérer la transparence et est compatible avec Flash 6 (sans transparence : Sorenson Spark) ou 8 (avec transparence : On2 VP6) et les versions ultérieures. Le F4V, compatible Flash 10, permet de gérer une image en haute définition.

Dans ce chapitre, nous abordons la vidéo composite, donc, FLV, pour en savoir plus sur la haute définition, reportez-vous au chapitre suivant.

Pour exporter la vidéo avec sa transparence ou pour Flash 7, cochez l'option FLV (voir Figure 6.25).

Figure 6.25

Multiplexeur.



Mécanisme du codec On2 VP6. Le codec On2 VP6, par rapport au codec Sorenson Spark, floute les artefacts générés par une compression de type Sorenson. Ainsi, l'image résultante, du fait que l'image est en mouvement, paraît plus propre que celle initialement obtenue avec le premier algo-

rithme, pour un poids égal. Mais le codec Sorenson Spark demeure adapté à certains types de vidéo pour lesquelles les mouvements restent moins prononcés. Le Sorenson Spark, dans ce cas, peut même offrir une meilleure compression que le On2VP6. Pensez à tester différents rendus selon le type de vidéo à publier avant de déterminer votre choix. La qualité de la compression étant assujettie à la nature intrinsèque de chaque vidéo, les résultats obtenus peuvent varier d'une vidéo à l'autre.

Vidéo

L'onglet Vidéo donne accès aux réglages détaillés de compression (voir Figure 6.26).

Figure 6.26

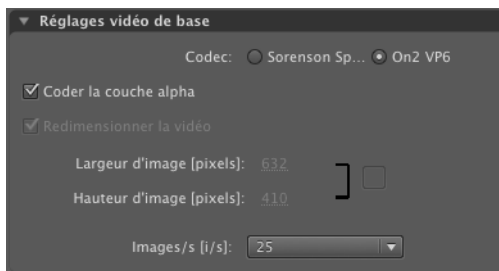
Vidéo (FLV).



- La première étape du réglage affecte les propriétés vidéo de base (voir Figure 6.27).

Figure 6.27

Réglages vidéo de base (FLV).



- Choisissez le codec Sorenson Spark pour un format de vidéo compatible avec Flash 6 et si le fichier vidéo provient d'une captation stable.
- Choisissez le codec On2 VP6 pour une vidéo compatible avec Flash 8 et les versions ultérieures. Ce format est aussi léger que le Sorenson mais lisse les artefacts rencontrés lors de la compression pour les images en mouvement. L'image obtenue est donc de

meilleure qualité pour un poids similaire dans les vidéos animées. Ce format autorise en outre la gestion de la transparence. En choisissant le codec On2 VP6, vous accédez à l'option Coder la couche alpha.

- Cochez l'option Coder la couche alpha pour que la transparence du fichier vidéo à encoder soit préservée dans l'échantillonnage. À défaut de cocher cette option, la transparence sera convertie en noir et, une fois importé dans Flash, vous ne pourrez plus voir les éléments disposés à l'arrière-plan de la vidéo.

Plus bas, nous distinguons les options de redimensionnement. Le redimensionnement interagit dynamiquement avec les options du menu Réglage du recadrage, situé à gauche de la fenêtre, dans l'onglet Sortie. Selon l'option choisie dans ce menu, le redimensionnement pourra apparaître inactif. Évitez bien sûr d'agrandir une vidéo. Vous en altéreriez le contenu.

- Pour changer la taille de la vidéo, en largeur et en hauteur, modifiez directement les valeurs affichées.
- Cliquez éventuellement sur la case située à droite des valeurs pour contraindre les modifications à un redimensionnement homothétique.

La cadence de la vidéo (ou fréquence) peut être définie dans le menu Images/s [i/s]. Utilisez une cadence élevée pour préserver la qualité de la vidéo, mais au détriment de son poids. Par défaut, la cadence est affichée sur 30ips, qui correspond à une cadence NTSC américaine. Portez la valeur de préférence sur 25ips pour être en conformité avec les standards européens PAL.



Conflit entre la cadence des vidéos et la cadence des animations Flash. Rappel : Les propriétés d'un document Flash affichent une cadence d'images. Si vous importez une vidéo, physiquement, dans un document Flash qui n'affiche pas la même cadence que celle de la vidéo, la vidéo sera désynchronisée par rapport à l'animation et par rapport à la bande son qui lui est propre. Si vous souhaitez intégrer physiquement une vidéo dans le scénario de Flash, utilisez la même cadence d'images pour les deux fichiers. Cette constatation ne vaut pas pour les vidéos gérées dynamiquement, *via* le composant FLVPlayback ou directement en ActionScript à l'aide de la classe NetStream.

Modifier la cadence des images. Certains logiciels, et notamment Motion, disposent d'un moteur de conversion très performant particulièrement adapté pour le remappage temporel (calcul des images ajoutées ou supprimées suite à la modification de durée d'un flux vidéo). Évitez, en règle générale, de modifier la cadence de l'image dans l'encodeur Adobe qui ne gère pas de recalcul sur des images intermédiaires aussi efficacement que dans des logiciels dédiés. Préférez gérer cette modification au sein même du logiciel de compositing pour un meilleur rendu.

La deuxième étape affecte les réglages de débit (voir Figure 6.28).

Plus bas dans la fenêtre, deux options d'encodage du débit sont proposées : CBR (*Constant Bite Rate* pour échantillonnage constant) et VBR (*Variable Bite rate* pour échantillonnage variable).

L'encodage en VBR désigne une compression qui évolue dans le temps (variable) en fonction du mouvement et de la richesse des images. Tandis qu'en CBR, la compression reste constante quel que soit le mouvement des images. Il est recommandé de choisir une com-

Dimensions standard d'une vidéo pour le Web

La taille d'une vidéo peut être gérée en fonction de la taille de votre document Flash ou inférieure, si sa vocation est de se distinguer d'un formatage classique, pour une création graphique par exemple. Mais dans le contexte de la gestion de vidéos aux formats standard, dans le cadre de la diffusion de séquence de reportage ou de fiction simple par exemple, voici les dimensions généralement observées :

Les tailles d'image standard pour une vidéo au format 4/3 de source PAL sont :

- Modem (56 Kbits/s) : 180×144
- ADSL : 360×288
- Câble : 576×460
- Câble/réseau d'entreprise, fibre optique : 720×576

Les tailles d'image pour une vidéo au format 16/9 PAL sont :

- Modem (56 Kbits/s) : 320×180
- ADSL : 540×304 ou 576×324
- Câble : 960×540 ou 1024×576
- Câble/réseau d'entreprise, fibre optique : 1920×1080

Les tailles d'image standard pour une vidéo au format 4/3 de source NTSC sont :

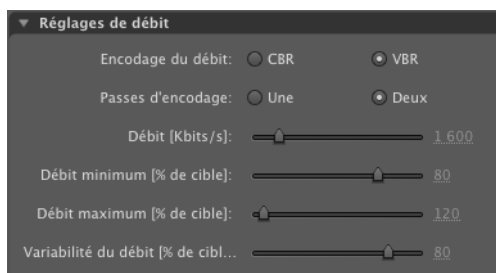
- Modem (56 Kbits/s) : 160×20
- ADSL : 320×240
- Câble : 512×384
- Câble/réseau d'entreprise, fibre optique : 640×480

Les tailles d'image pour une vidéo au format 16/9 NTSC sont :

- Modem (56 Kbits/s) : 192×108
- ADSL : 384×216
- Câble : 448×252
- Câble/réseau d'entreprise, fibre optique : 704×396

Figure 6.28

Réglages de débit (FLV).



pression VBR pour un rendu de meilleure qualité, mais au risque que certains passages de la vidéo s'arrêtent pendant la lecture. Le mode CBR est préconisé en revanche pour des flux plus homogènes et risque moins d'interrompre la lecture des séquences vidéo, mais les images en mouvement seront dégradées car elles sont toutes compressées à l'identique, que les images soient riches et animées ou pauvres et fixes. La qualité des animations riches risque donc d'en pâtir de manière perceptible. L'encodage VBR sollicite davantage, de part la nature fluctuante de la compression, les ressources processeur du serveur qui met ces vidéos à disposition. Certains problèmes de synchronisation de l'audio ont également été observés dans des conditions extrêmes de bas débit. Pour notre exemple, restez sur VBR. Le codec Sorenson Spark ne permet enfin qu'une compression en CBR.

Le paramètre Passes d'encodage désigne le nombre de passages sur la vidéo que l'encodeur doit effectuer avant de déterminer la quantité d'informations à supprimer. Plus l'encodeur analyse la vidéo, meilleur en sera le rendu. Il est donc recommandé de choisir l'option Deux passes, même si cela augmente le temps de calcul pour l'encodage. Cochez l'option Deux.

Sous le nombre de passes, un réglage sur la compression est disponible. Les valeurs indiquent le débit pour lequel la vidéo doit être adaptée. Ce débit est défini en nombre de kilobytes par seconde et fait directement référence au débit dont les utilisateurs disposent avec leur connexion Internet pour lire la vidéo.

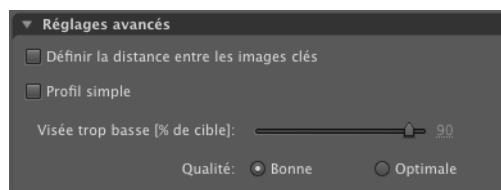
Si une passe a été activée, alors, un seul taux de compression, unique, est proposé pour l'ensemble de la vidéo. Si l'option Deux passes en revanche a été activée, deux taux sont proposés. Le taux maximum détermine le seuil de compression pour les images riches et mouvementées, alors que le seuil minimum détermine celui des images fixes. Pour éviter une rupture de flux lorsque le lecteur atteint une image animée et riche, réduisez l'écart entre les deux seuils proposés en rapprochant autant que possible le seuil maximum du seuil minimum. Deux seuils de valeur identiques équivalent à une compression Une passe. Vous augmentez simplement la durée de l'encodage.

Enfin, la variabilité du débit, affichée pour deux passes uniquement, permet de déterminer le moment où l'on considère que l'image doit basculer d'une compression à l'autre. Plus la valeur est élevée, plus la compression basculera d'un seuil à l'autre intempestivement. Pour un débit relativement plus constant, réduisez cette valeur, mais au détriment de la qualité des images animées et riches. Conservez pour notre exemple la valeur définie par défaut (80).

La troisième étape consiste à ajouter des réglages plus avancés (voir Figure 6.29).

Figure 6.29

Réglages avancés
(FLV).



D'abord, la distance entre les images-clés permet de définir quelles images de la vidéo seront codées intégralement. L'encodeur détermine automatiquement une image-clé toutes

les deux secondes pour une lecture simple du flux vidéo. Conservez dans cet exemple la valeur par défaut (automatique).



Les images-clés. En vidéo, la compression du signal consiste à éliminer les informations redondantes d'une image sur l'autre et ne conserver que les informations qui changent. Les images-clés servent à rafraîchir l'image courante avec une image pleine. Cela permet d'éviter certaines aberrations visuelles surtout lorsque l'on autorise l'utilisateur à naviguer, à l'aide de fonctionnalités d'accélération ou de chapitrage, à l'intérieur du flux vidéo. Plus la vidéo comporte d'images-clés, plus elle sera nette et plus la navigation pourra être ciblée, mais plus elle sera lourde également. Nous revenons plus en détail sur cette option au Chapitre 8.

Plus loin, le profil sert à optimiser la compatibilité de la vidéo avec des équipements faibles en carte vidéo. Si l'option est activée, la vidéo sera altérée et nécessitera alors moins de ressources sur la machine utilisateur. Ne cochez pas l'option Profil simple.

Le dernier réglage permet de définir le pourcentage de vidéo à précharger dans le cas où le débit serait vraiment trop faible, avant de pouvoir être lu automatiquement. Conservez la valeur par défaut.

Une option de qualité permet enfin de choisir entre deux valeurs. Bonne spécifie une meilleure qualité au détriment de la rapidité du chargement pour les connexions basses. Optimale, à l'inverse, rend la vidéo plus accessible mais au détriment de sa qualité.

Audio

La compression Audio (voir Figure 6.30), lorsqu'elle est requise, peut être optimisée en la passant en Mono. Vous pouvez également compresser le signal en réduisant la valeur du menu Débit. Notez que le son partage le débit avec la vidéo. Plus vous augmentez le débit pour le son, plus vous diminuez celui qui restera disponible pour l'image.

À titre indicatif, une valeur inférieure à 64 Kbits/s commence à affecter sérieusement la qualité sonore d'une musique. Une valeur inférieure à 32 Kbits/s commence à affecter de manière perceptible le son de la voix. Privilégiez un réglage généralement compris entre 64 et 128 Kbits/s grand maximum.

Il est possible d'encoder le son dans un format audio de meilleure qualité, mais uniquement avec le format F4V (voir Chapitre 7).

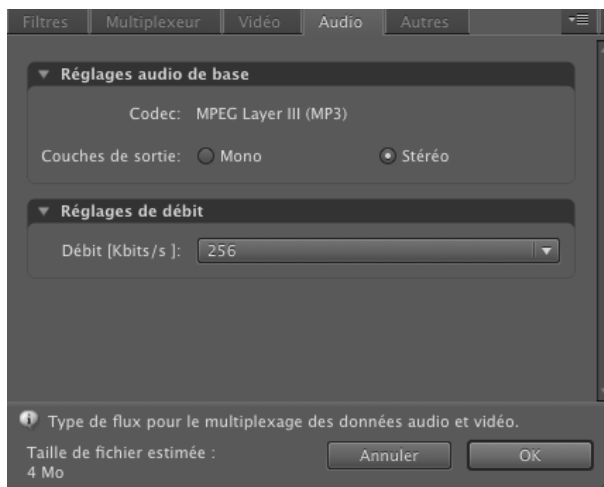
Autres

Un dernier onglet permet de publier directement la vidéo sur un serveur FTP, pour mettre à jour un podcast audio ou vidéo par exemple, sans avoir à utiliser de logiciel de transfert pour ce faire (voir Figure 6.31). Renseignez, dans ce cas précis, les codes FTP fournis par votre hébergeur pour procéder directement à une mise en ligne du flux vidéo sur un serveur distant.

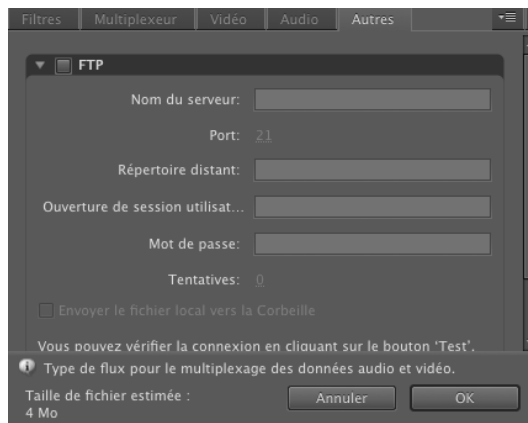
Une fois tous les réglages définis. Cliquez sur OK. La boîte de dialogue se ferme et, dans la fenêtre de départ, le fichier vidéo est prêt à être encodé. Cliquez sur le bouton Démarrer la file d'attente pour lancer le calcul.

Figure 6.30

Audio (FLV).

**Figure 6.31**

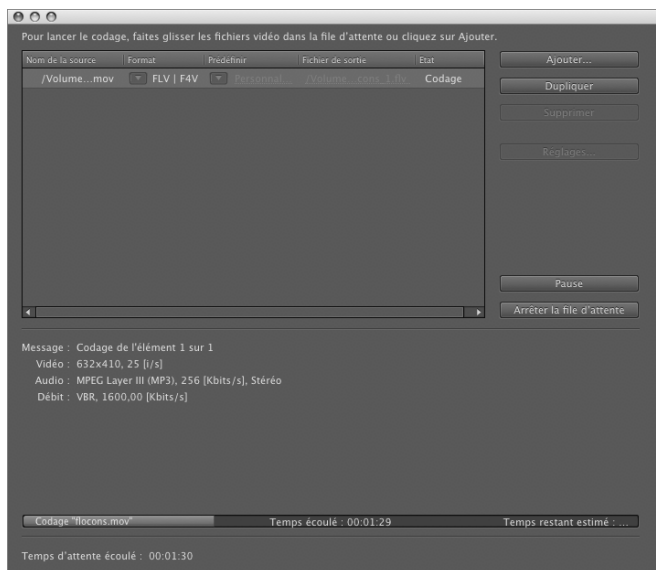
FTP.



Une jauge matérialise alors la progression de l'encodage (voir Figure 6.32). La compression se termine par un signal sonore et le fichier est enregistré à l'endroit spécifié. Par défaut, si vous ne spécifiez pas d'emplacement ni de nom de sortie, le fichier reprend le nom de la vidéo originale et est enregistré dans le même répertoire.

Figure 6.32

Rendu.



La vidéo peut maintenant être intégrée à Flash.

À retenir

- La compression audio-vidéo est un compromis entre les dimensions de l'image, le taux de compression de l'image, le taux de compression du son et le nombre d'images-clés.
- Il est possible d'exploiter le format FLV y compris pour diffuser de l'audio. Cela permet d'y associer, éventuellement, de l'interactivité.
- La cadence des images de la vidéo doit être identique à la cadence de la scène, dans flash, si la vidéo doit être importée physiquement dans le scénario. Ce n'est pas le cas pour les vidéos gérées avec un composant ou avec la classe NetStream.

Intégrer de la vidéo composite dans Flash

Il y a plusieurs manières d'intégrer de la vidéo dans Flash. Nous pouvons naturellement la traiter *via* ActionScript, mais aussi en utilisant un composant préprogrammé. Dans cette section, nous utilisons le composant FLVPlayback. Nous abordons la gestion de la vidéo en ActionScript au Chapitre 8.



Exemples > ch6_videoComposite_1 fla

Dans le document "ch6_videoComposite_1 fla", seul le calque `fond_mc` affiche un contenu. Un autre calque, nommé `video`, demeure vide.

Dans ce document, nous allons placer sur la scène une occurrence du composant vidéo FLV-PlayBack et, à travers l'Inspecteur de composants, nous allons établir une liaison entre cette occurrence et la vidéo exportée au format FLV.

1. Affichez la fenêtre des composants (Fenêtre > Composants) – voir Figure 6.33.

Figure 6.33

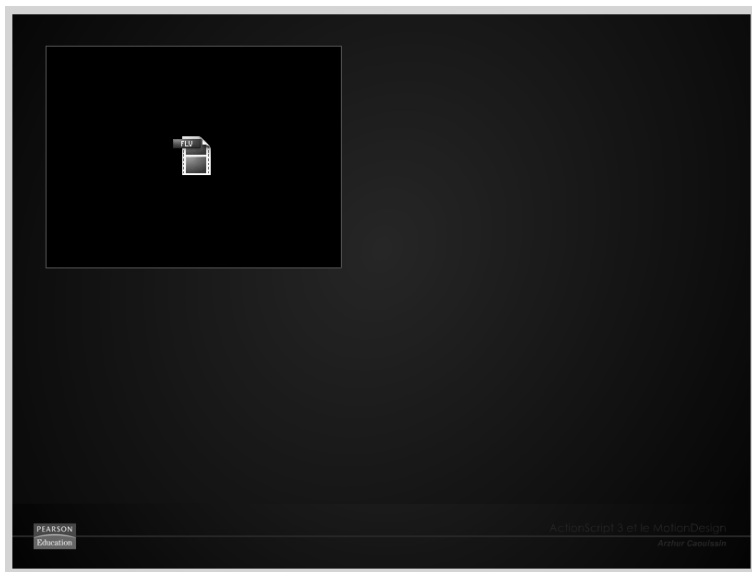
Fenêtre des composants.



2. Dans la catégorie Video, glissez-déposez l'élément intitulé FLVPlayback directement sur la scène (voir Figure 6.34).

Figure 6.34

Composant placé sur la scène.



3. Puis, affichez la fenêtre Inspecteur de composants (Fenêtre > Inspecteur de composants) – voir Figure 6.35.

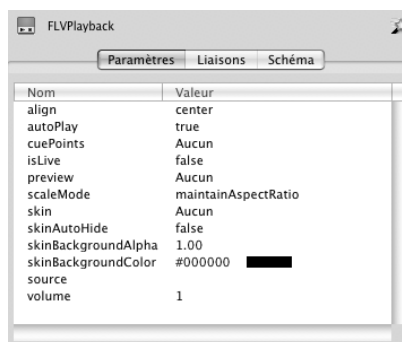
Pour visualiser les options d'un composant depuis la fenêtre Inspecteur de composants, le composant doit être préalablement sélectionné sur la scène. Cliquez au besoin sur le composant vidéo placé dans la scène pour le sélectionner si celui-ci n'était pas actif.

Composant FLVPlayback pour ActionScript 2 ou 3

Lorsque vous placez un composant sur la scène, le code utilisé pour le créer n'est pas le même selon que vous publiez un document en ActionScript 2 ou 3. Aussi, les paramètres personnalisables de la fenêtre Inspecteur de composants n'affichent pas les mêmes noms selon le contexte de développement. Le paramètre source appartient à un codage ActionScript 3. En ActionScript 2, ce paramètre est identifié sous le terme ContentPath.

Figure 6.35

Fenêtre Inspecteur de composants.



Dans la fenêtre des composants, sélectionnez le paramètre intitulé source. Puis, dans le champ de texte de saisie, cliquez à droite pour afficher une icône qui représente une loupe. Cliquez sur la loupe pour ouvrir une boîte de dialogue de sélection de fichier. Dans cette boîte de dialogue, cliquez sur l'icône qui représente partiellement un dossier, située à droite, pour ouvrir enfin la fenêtre de sélection de fichier. Sélectionnez alors, sur votre poste de travail, la vidéo enregistrée au format FLV. L'option Identique aux dimensions sources permet de modifier les dimensions du composant et l'adapter aux dimensions de la vidéo appelée en référence.

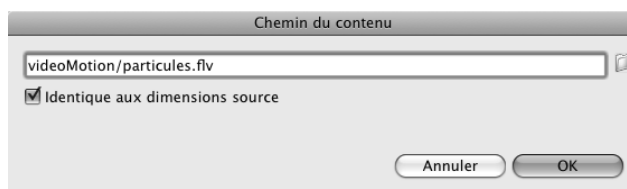


Avertissement pour les utilisateurs de Windows. Attention, sous Windows, l'icône de sélection de fichier du composant FLVPlayback n'est presque pas visible. Vous devez cliquer à l'extrémité du bord droit de la boîte de dialogue pour l'activer. Le cas échéant, saisissez manuellement le chemin relatif qui relie le document Flash à la vidéo (voir Figure 6.37).

Puis refermez la fenêtre en validant les étapes (voir Figure 6.36).

Figure 6.36

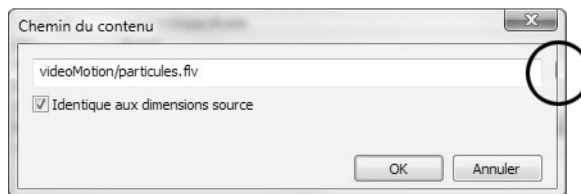
Chemin du contenu,
sous Macintosh.



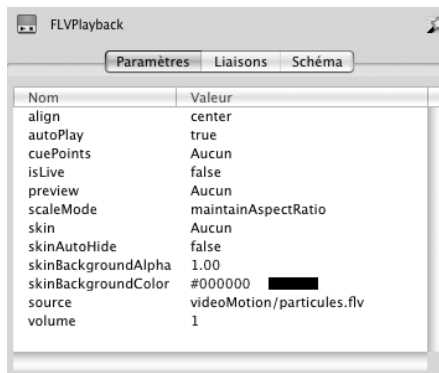
Une fois la liaison activée, la fenêtre Inspecteur de composants affiche le chemin d'accès au fichier vidéo FLV (voir Figure 6.38).

Figure 6.37

Chemin
du contenu,
sous Windows.

**Figure 6.38**

Chemin
enregistré dans la
fenêtre Inspecteur
du composant.



Paramètre source des composants vidéos. Vérifiez toujours que le chemin défini pour le paramètre source des composants vidéos part bien de la position relative de la page HTML qui contient le document Flash (ou du Flash lui-même). Dans certains cas, si vous travaillez en réseau par exemple, Flash peut enregistrer un chemin absolu qui part de la racine de votre système en ciblant votre réseau local. Un chemin de ce type ne serait pas fonctionnel une fois le projet en ligne. Il faut donc toujours vérifier l'emplacement désigné une fois l'option validée. Le chemin doit être relatif. Par exemple, le chemin "videos/mavideo.flv" peut être valide si la vidéo se nomme "mavideo.flv" et qu'elle se trouve dans un dossier nommé "videos", situé au même niveau que votre document Flash ou du moins, au même niveau que la page HTML qui affiche votre document Flash. En revanche, un chemin du type "Disque/Poste de travail/projet/site/videos/mavideo.flv" ne sera pas valide dans ce contexte.

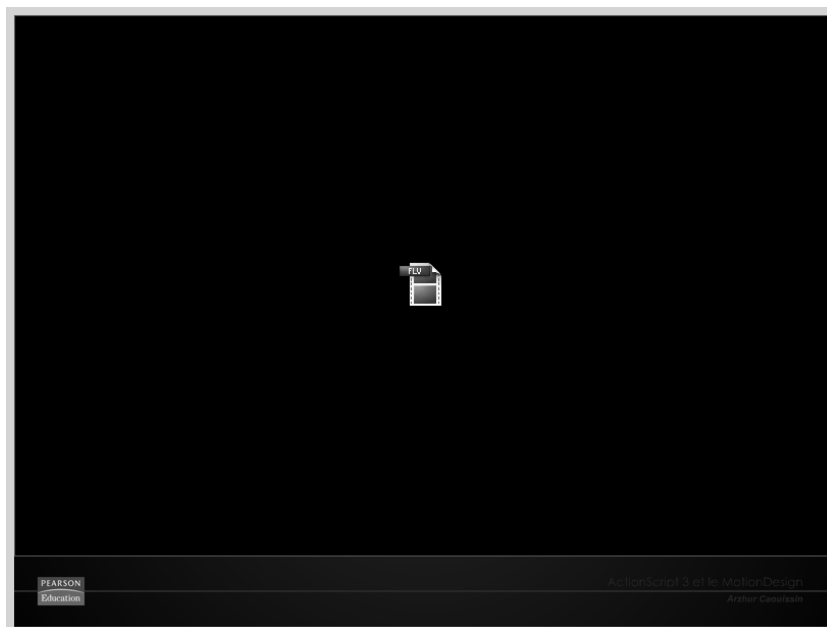
Dès la fermeture de la boîte de dialogue, le composant est automatiquement adapté aux dimensions de la vidéo. Au besoin, repositionnez ce composant à l'origine de la scène, ou à l'emplacement voulu (voir Figure 6.39).

Selon la configuration de votre application, une console de lecture peut être attachée automatiquement à votre composant vidéo. Elle permet de piloter la vidéo à la publication. Nous aborderons ces fonctionnalités dans le prochain chapitre. Pour une vidéo composite, qui apparaît généralement comme un élément purement graphique, sans outils de contrôle autres que l'interactivité développée par ailleurs, nous choisissons de masquer cette console.

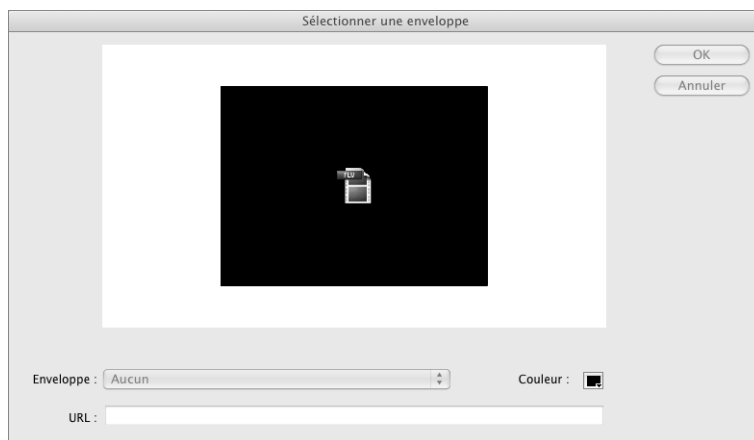
1. Pour masquer la console de lecture affichée par défaut, dans l'Inspecteur de composants, cliquez à droite de l'option Skin jusqu'à ouvrir une boîte de dialogue (voir Figure 6.40).
2. Dans cette fenêtre, vous pouvez choisir le type d'habillage pour la console.
3. Dans le menu Enveloppe, sélectionnez l'option Aucun. Puis, validez.

Figure 6.39

Repositionnement éventuel du composant.

**Figure 6.40**

Suppression des contrôles de lecture



La vidéo ne possède plus de contrôle de lecture. En publiant le document, le flux FLV joue l'animation de particules (feu d'artifice) et laisse apparaître la scène du document Flash située en arrière-plan (voir Figure 6.41).

Si vous souhaitez utiliser une Skin pour contrôler la lecture de la vidéo, reportez-vous au Chapitre 8 pour le détail de ces options.

En publiant le document Flash, la vidéo n'est jouée qu'une seule fois. Il est possible de créer des boucles ou de contrôler plus précisément la lecture de la vidéo. Nous abordons également ces notions dans les deux chapitres suivants.

Figure 6.41

Aperçu après
publication



Le langage

Paramétrer un composant *via* ActionScript

L'ensemble des propriétés du composant FLVPlayback peut être géré en ActionScript. Pour cela, il suffit d'invoquer le nom d'occurrence du composant et d'y attacher la propriété à modifier, à l'aide de la syntaxe pointée. Pour modifier la source dynamiquement, inscrivez par exemple : `maVideo.source="videos/film2.flv"`. Nous abordons en détail certaines de ces propriétés au Chapitre 8.

Vous pouvez aussi ajouter un aperçu de la vidéo pour faciliter sa manipulation dans la scène :

1. Pour ajouter un aperçu, dans l'Inspecteur de composants, cliquez à droite du paramètre Preview.
2. Dans la nouvelle boîte de dialogue, arrêtez la vidéo à l'emplacement voulu.
3. Puis, cliquez sur OK.

Un aperçu matérialise à présent la vidéo dans la scène. Cet aperçu n'apparaît pas à la publication. Il ne sert que pour manipuler la vidéo dans Flash. Cet aperçu étant une image capturée en JPEG, il ne gère pas la transparence. Mais cela n'affecte en rien le rendu final, obtenu à la publication du document.



Reconstituer un univers 3D avec une vidéo aplatie. Flash CS4 possède un moteur d'affichage simili 3D. Il est possible d'animer dans l'espace, des symboles de type MovieClip, avec des propriétés de rotation et de position. Un composant vidéo est un objet qui peut être contenu dans un symbole de type MovieClip. Il est donc possible d'animer et projeter des vidéos dans l'espace 3D de Flash.

Dans le cadre d'une animation de particules publiée au format FLV avec sa couche transparente, vous pouvez donc recréer un effet de particules en 3D dans Flash. Pour cela, créez un document compre-

nant plusieurs occurrences de MovieClip identiques, chacune répartie sur un calque distinct et de propriété de position et de rotation 3D différenciées. Mais elles sont toutes centrées et superposées dans la page, de sorte que les contenus se croisent au centre du document. Placez, dans le symbole de la bibliothèque ou dans une des occurrences de la scène, le composant FLVPlayback. Associez sa source à un flux vidéo FLV de particules codé avec la transparence.

En publiant l'animation, vous pouvez voir que la vidéo, *a priori* aplatie, mais redistribuée spatialement, et avec sa transparence, adopte une toute nouvelle dimension. Le fichier "ch6_videoComposite_3 fla" propose ce type de mise en forme.

La transparence, associée à l'affichage 3D de Flash, ouvre des perspectives d'exploitation des flux vidéo, mais requiert, bien sûr, un minimum de ressources machine – ce qui réserve ce type d'agence-ment à des configurations solides.

À retenir

- Pour intégrer une vidéo FLV dans Flash, nous utilisons le composant FLVPlayback.
- Ce composant présente des options différentes selon la version de langage utilisée pour la publication du document.

Synthèse

Dans ce chapitre, vous avez appris à intégrer des vidéos standard dans un document Flash, à transposer des créations vidéos composites avec de la transparence dans un document Flash. Vous avez appris à optimiser un flux vidéo pour le Web et à identifier les contraintes et apports du format FLV pour une utilisation la plus qualitative possible. Vous êtes en mesure de réaliser des interfaces de sites riches qui mixent les flux vidéo avec des contenus graphiques traditionnels en Flash.

7

La vidéo HD en F4V

Introduction

Depuis la version 10 (CS4), Flash accepte le codec H-264 à travers le format vidéo F4V. Ce codec offre une compression d'une excellente qualité où les artefacts demeurent presque imperceptibles. Cette prouesse est telle que l'on peut désormais déployer une vidéo dans un format étendu (haute résolution) sans perdre en qualité d'affichage.

Cependant, à la différence du format FLV, le format F4V ne gère pas la couche transparente (canal alpha). Le F4V convient donc plus particulièrement à la diffusion de séquences vidéo isolées et non composites (extraits de films, introductions, transitions, décors d'arrière-plan animés, interviews, infographies animées).

Dans ce chapitre, nous allons voir comment encoder au format F4V et distribuer des vidéos de haute qualité vers Flash, y compris vers des versions du lecteur *a priori* incompatibles avec ce format. Nous aborderons également quelques astuces qui permettent de compresser confortablement une vidéo de grande taille sans perdre sur la qualité du rendu.

Pour l'ensemble de ces exemples, nous utilisons des créations originales réalisées par la société gKaster, spécialisée dans le Motion Design, et mises amicalement à notre disposition pour cette démonstration (www.gKaster.com).

À l'issue de ce chapitre, vous serez en mesure d'intégrer des vidéos de grande qualité au sein de documents Flash, y compris d'anciennes générations.

Encoder en F4V avec Adobe Media Encoder

Pour créer un fichier F4V, vous pouvez l'échantillonner avec Adobe Media Encoder. Dans cette section, nous abordons uniquement les réglages spécifiques au format F4V qu'apporte ce logiciel. Si vous voulez découvrir les réglages de base pour la compression d'une vidéo pour Flash, communs au format FLV et F4V, reportez-vous au chapitre précédent, à la section "Échantillonner la vidéo pour Flash".



Exemples > gKaster > gKaster-amusement.mov

Dans cette section, nous détaillons les options d'encodage pour le format F4V, avec le codec H-264.

1. Lancez l'application Adobe Media Encoder.
2. Ajoutez, dans la file de rendu, le fichier nommé gKaster-amusement.mov, disponible dans le dossier gKaster des exemples du livre (voir Figure 7.1).

Figure 7.1

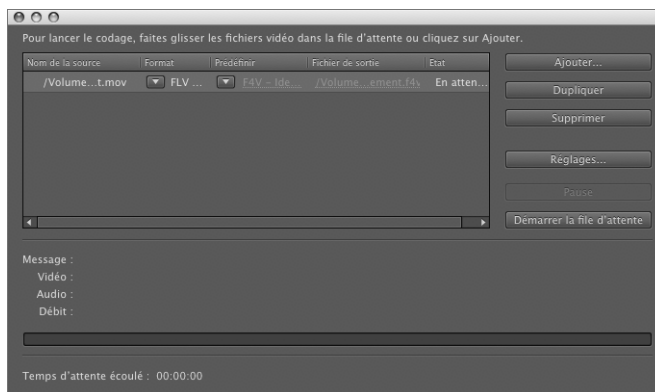
Aperçu de la vidéo gKaster-amusement.mov.



3. Dans la fenêtre de l'encodeur Adobe, cliquez directement sur le lien jaune de la colonne Prédéfinir pour accéder aux réglages personnalisés (voir Figure 7.2).

Figure 7.2

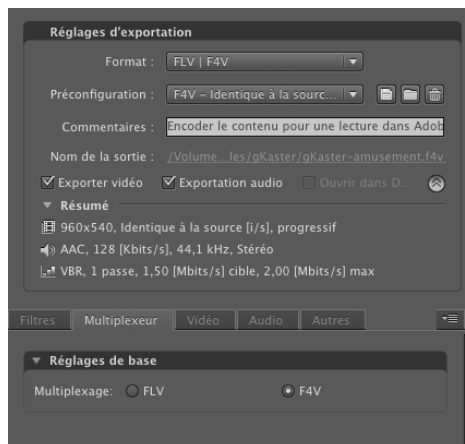
Fenêtre Adobe Media Encoder.



4. La fenêtre de réglages s'ouvre. Dans la partie droite de la fenêtre, activez l'onglet Multiplexeur pour définir le type d'encodage Flash (voir Figure 7.3).

Figure 7.3

Fenêtre Réglages d'exportation, onglet Multiplexeur.



5. Sélectionnez l'option F4V si celle-ci n'est pas déjà active.
6. Puis, cliquez sur l'onglet Vidéo pour définir les réglages relatifs au format F4V (voir Figure 7.4).

Figure 7.4

Onglet Vidéo.



Dans la catégorie Vidéo, nous découvrons des options légèrement différentes de celles affichées pour le format FLV.

Onglet Vidéo

Le document Flash mesure 800 pixels de large. La vidéo gKaster en fait 960. Nous allons donc commencer par redimensionner cette vidéo pour la contenir dans notre document.

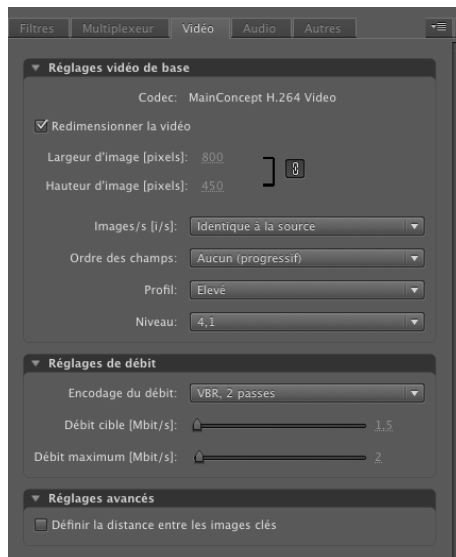
1. Cochez d'abord l'option de redimensionnement homothétique située à droite des champs Largeur et Hauteur.
2. Puis, inscrivez la valeur 800, dans le champ relatif à la largeur. La valeur Hauteur s'adapte automatiquement (voir Figure 7.5).

Dans la partie Réglages vidéo de base, vous trouverez de nouveaux paramètres, spécifiques au format F4V.

Le format F4V utilise le codec H-264 (ou MPEG-4 Partie 10/AVC). Ce codec est développé en commun par le VCEG (*Video Coding Expert Group* ou groupe d'experts en encodage vidéo), l'ITU-T (*International Telecommunications Union, Telecommunications Standardization Sector* ou le département standardisation de l'union internationale des télécommunications) et le MPEG (*Movie Picture Expert Group* ou le groupe d'experts en images animées). Il se présente comme une amélioration du codec MPEG-2 et MPEG-4 et repose sur le principe suivant.

Figure 7.5

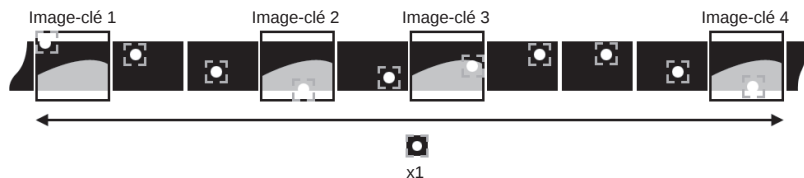
Réglages vidéo
de base (F4V).



Lorsqu'une compression classique n'utilise qu'une image-clé de référence pour définir les zones qui changent d'une image sur l'autre (codage différentiel), le H-264 peut utiliser jusqu'à quatre images de référence. Cette logique lui permet de détecter le mouvement global, sur la durée, de groupes de pixels entiers, et de ne coder que l'emplacement de ces groupes et non la teneur colorimétrique de chaque pixel qui compose chaque groupe. Ainsi, quand une scène se répète à l'intérieur d'une vidéo, ou si plusieurs événements graphiques identiques sont reproduits ou en mouvement sur différentes images-clés, ces événements ne sont codés qu'une fois (voir Figure 7.6). Cela permet de compacter largement les vidéos longues aux effets répétitifs, ou dont la nature des images reprend toujours le même type d'informations, qu'elles soient mobiles ou non (un sujet frontal sur un fond blanc, même en mouvement, sera peu gourmand en poids, par exemple). Une compression H-264 appliquée à une vidéo perd ainsi environ 80 % de son poids lorsqu'une vidéo compressée en MPEG-4 Partie 2 n'en perd que 50 %, et sans différence de qualité entre les deux modes de compression.

Figure 7.6

Représentation
du principe de
l'algorithme
H-264.



Les blocs, qui fondent la base de l'algorithme H-264, sont constitués de 16 pixels (4×4). Outre l'avantage observé sur le poids du fichier, cette technique permet également de faciliter et stabiliser la lecture du flux vidéo surtout lorsqu'elle est diffusée en ligne. Car chaque paquet peut ainsi être lu indépendamment des autres et contribue à remplir automatiquement les trous résiduels éventuellement obtenus en cas de rupture de flux.

Ces blocs sont utilisés pour définir les images-clés de référence. Si une portion de l'image se *répète* ailleurs, l'algorithme identifie les nombreuses occurrences *identiques* de la zone

de base et la reproduit virtuellement, même dans une image-clé. Cela allège donc aussi le poids des images-clés.

Le H-264 offre enfin un éventail de compression suffisamment large pour être employé pour des flux de haute résolution, avec une fréquence d'image élevée (diffusion HD), aussi bien que pour des flux très réduits et de faible définition (appareils mobiles).

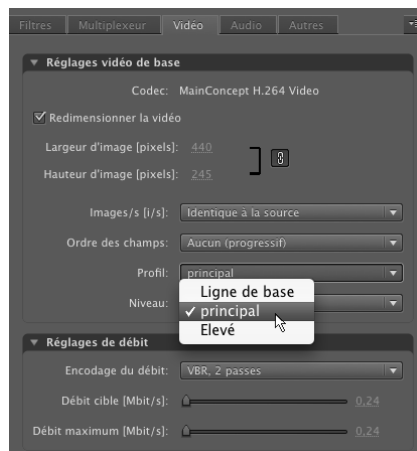
De nombreuses options de compression sont possibles pour le H-264. Deux options essentielles sont disponibles dans l'encodeur Adobe : le profil et le niveau.

Profil

Le codec H-264 possède initialement sept profils de compression permettant de coder les images de faible résolution aux images à haute résolution. Les profils disponibles sont de qualité croissante et à sélectionner en fonction de la dimension de la vidéo. Parmi ces sept profils propres au H-264, Adobe Media Encoder en propose trois : ligne de base, principal et élevé (voir Figure 7.7).

Figure 7.7

Sélection
d'un profil.



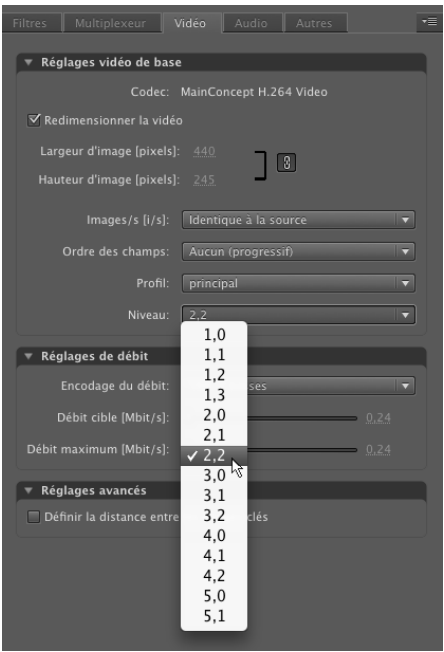
- **Ligne de base.** Ce profil est adapté aux contenus vidéos à destination d'appareils à faibles ressources (appareils nomades, mobiles, visio-conférence, caméras réseau). Le temps de latence de ce profil (temps requis pour compresser et décompresser le signal) est faible. Ce profil est donc tout particulièrement adapté pour l'encodage en direct de mouvements rapides (zoom, inclinaisons, vues panoramiques).
- **Principal.** Ce profil est adapté aux contenus à destination du Web, de la diffusion en vrai et faux streaming de contenus déjà enregistrés. Ce profil possède des capacités de robustesse à la perte de données qui en fait l'option idéale pour la vidéo diffusée en ligne.
- **Élevé.** Ce profil est utilisé pour la diffusion et le stockage sur disque, l'encodage Blue-Ray, HD-DVD et la télévision haute définition française. Il convient également à la diffusion de contenu en ligne de très haute qualité, pour lesquels on prévoit un chargement de la vidéo avant d'en activer la lecture, pour la V.O.D. par exemple.

Niveau

Selon les dimensions de la vidéo, nous déterminons un niveau (*Level*) sur une échelle de 1 à 5.1. Plus l’image à encoder est grande, plus le niveau doit être élevé (voir Figure 7.8).

Figure 7.8

Sélection d’un niveau



Dans le Tableau 7.1, la valeur de niveau à sélectionner est définie en fonction de la résolution de l’image et sa cadence. Pour notre exemple, conservez les valeurs par défaut.

Tableau 7.1 : Niveaux en H-264 (extrait de la page <http://fr.wikipedia.org/wiki/H.264#Levels>)

Niveau	Nombre maximum de macro-blocs lus par seconde	Débit maximum en bits pour les profils Ligne de base et Principal	Débit maximum en bits pour le profil Élevé	Exemples de définition et d’images par seconde par niveau
1	1485	64 Kbit/s	80 Kbit/s	128 × 96/30.9 176 × 144/15.0
1b	1485	128 Kbit/s	160 Kbit/s	128 × 96/30.9 176 × 144/15.0
1.1	3000	192 Kbit/s	240 Kbit/s	176 × 144/30.3 176 × 240/10.0
1.2	6000	384 Kbit/s	480 Kbit/s	176 × 144/60.6 320 × 240/20.0 352 × 288/15.2
1.3	11880	768 Kbit/s	960 Kbit/s	352 × 288/30.0
2	11880	2 Mbit/s	2.5 Mbit/s	352 × 288/30.0

Tableau 7.1 : Niveaux en H-264 (extrait de la page <http://fr.wikipedia.org/wiki/H.264#Levels>) (suite)

Niveau	Nombre maximum de macro-blocs lus par seconde	Débit maximum en bits pour les profils Ligne de base et Principal	Débit maximum en bits pour le profil Élevé	Exemples de définition et d'images par seconde par niveau
2.1	19800	4 Mbit/s	5 Mbit/s	352 × 480/30.0 352 × 576/25.0
2.2	20250	4 Mbit/s	5 Mbit/s	720 × 480/15.0 352 × 576/25.6
3	40500	10 Mbit/s	12.5 Mbit/s	720 × 480/30.0 720 × 576/25.0
3.1	108000	14 Mbit/s	17.5 Mbit/s	1 280 × 720/30.0 720 × 576/66.7
3.2	216000	20 Mbit/s	25 Mbit/s	1 280 × 720/60.0
4	245760	20 Mbit/s	25 Mbit/s	1 920 × 1 080/30.1 2 048 × 1 024/30.0
4.1	245760	50 Mbit/s	62.5 Mbit/s	1 920 × 1 080/30.1 2 048 × 1 024/30.0
4.2	522240	50 Mbit/s	62.5 Mbit/s	1 920 × 1 080/64.0 2 048 × 1 088/60.0
5	589824	135 Mbit/s	168.75 Mbit/s	1 920 × 1 080/72.3 2 560 × 1 920/30.7
5.1	983040	240 Mbit/s	300 Mbit/s	1 920 × 1 080/120.5 4 096 × 2 048/30.0

Onglet Audio

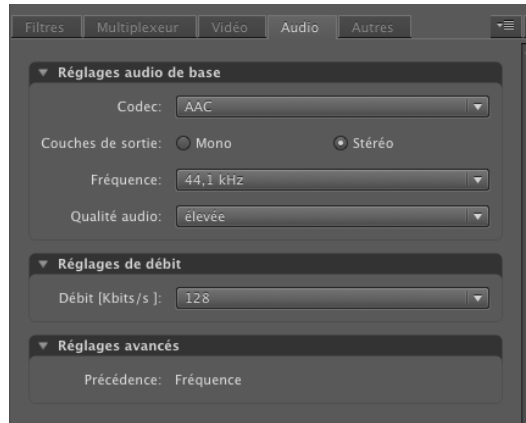
Le format F4V offre une compression audio de bien meilleure qualité que le FLV. Le codec AAC, supérieur à la qualité du MP3, propose une profondeur de son plus étendue, proche de celle d'un CD audio, mais naturellement, de poids légèrement plus élevé. C'est aussi le codec employé par Apple et RealNetWorks pour la diffusion de leurs contenus audios. Pour contrôler la compression audio, nous considérons véritablement deux paramètres : la fréquence et le débit (voir Figure 7.9).

Fréquence. La fréquence indique le nombre de fois qu'une ponction d'échantillons sonores est effectuée par seconde (à ne pas confondre avec la longueur d'onde qui représente la sonorité elle-même). Plus la fréquence est élevée, plus l'on peut distinguer les variations du son sur une durée donnée. Mais plus cela augmente nécessairement le poids du fichier.

Débit. Le débit consiste à synthétiser les informations en les arrondissant aux valeurs numériques les plus proches. Plus la compression du débit est forte, plus la représentation du son est grossière et perd en nuances, même avec une fréquence élevée.

Figure 7.9

Réglages Audio
(F4V).



Les paramètres de réglage affichés sont ceux définis par défaut pour le préréglage à partir duquel nous effectuons cette compression. Ces préréglages sont ceux affichés dans le menu Préconfiguration, au sommet de la fenêtre d'encodage, ou bien dans la liste Prédéfinir, de la file d'attente de rendu. Selon le type de préréglage choisi, vous n'observerez pas, par conséquent, les mêmes taux de compression.

1. Conservez pour cet exemple les valeurs par défaut.
2. Puis, refermez la boîte de dialogue des réglages de compression en cliquant sur OK.
3. Lancez le rendu en activant le bouton Démarrer la file d'attente.
4. Puis, quittez l'application.

Seuls des tests de rendu effectués en réel vous donneront une idée du résultat obtenu. N'hésitez pas à dupliquer le réglage dans la file d'attente de rendu et à y appliquer différents paramètres de compression audio pour ne retenir que celui qui vous convient le mieux.

À retenir

- Le format F4V offre une grande souplesse de compression qui permet d'adapter la vidéo à tout type de support, d'une projection HD à une image réduite pour appareil mobile, sans perte de qualité.
- Le format F4V ne permet pas la gestion de la transparence (couche alpha).
- L'encodage audio, en F4V, est supérieur en qualité au codage MP3 du format FLV.
- Si nous intégrons la vidéo avec un composant FLVPlayback, le format F4V requiert un lecteur Flash récent.

Encodage F4V avec Quick Time

Cela fait déjà longtemps qu'il est possible d'exporter avec une compression H-264 depuis Quick Time. Le format vidéo F4V de Flash qui repose sur le codec H-264, peut donc aussi être généré depuis un fichier Quick Time et être intégré à un composant Flash FLVPlayback, il sera normalement interprété. Pour éviter les messages possibles d'avertissement et de sécurité de Flash, lors de l'intégration de la vidéo, songez à substituer l'extension

obtenue avec Quick Time par .f4v. Mais, même sans en modifier l'extension, le fichier pourra néanmoins être lu comme un fichier F4V.



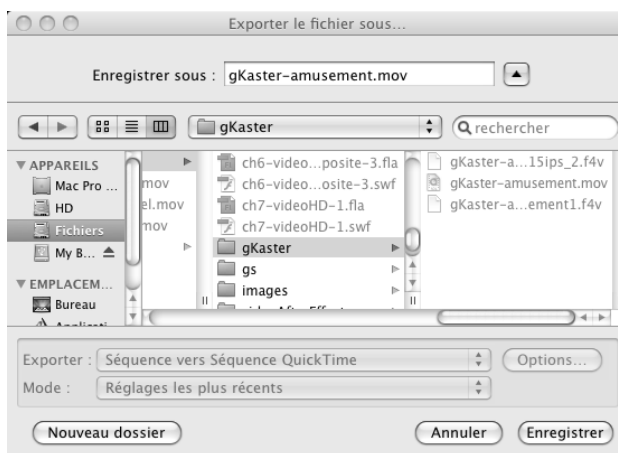
Exemples > gKaster > gKaster-amusement.mov

Dans cette section, nous détaillons le moyen d'encoder une vidéo, à partir de Quick Time, au format F4V, pour Flash.

1. Pour exporter la vidéo au format H-264 avec Quick Time, dans Quick Time, faites Fichier > Exporter.
2. Dans la boîte de dialogue d'enregistrement, cliquez sur le bouton options (voir Figure 7.10).

Figure 7.10

Fenêtre d'exportation Quick Time.



3. Dans la boîte de réglages, cliquez sur Réglages de la catégorie Vidéo (voir Figure 7.11).

Figure 7.11

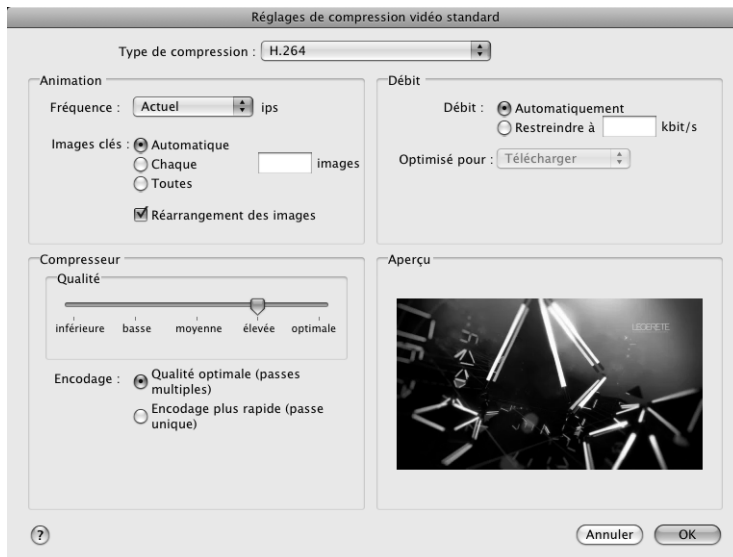
Réglages de la séquence.



4. Puis, dans la fenêtre d'encodage, dans le menu Type de compression, sélectionnez H-264.
5. Dans les options de réglages pour le type H-264, choisissez Passes multiples et une qualité élevée, ainsi qu'un nombre d'image-clés automatique (voir Figure 7.12). Puis, validez.

Figure 7.12

Compression
H-264.



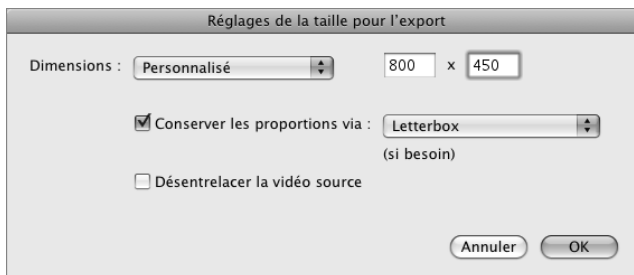
Puisque le document Flash dont nous disposons mesure 800 pixels de large, nous pouvons également redimensionner ici la vidéo en activant l'option taille :

1. Cliquez sur le bouton Taille.
2. Dans la nouvelle boîte de dialogue, spécifiez une taille personnalisée. Puis, dans les champs situés à droite, inscrivez manuellement les valeurs 800 et 450.

Les options de proportions permettent de rogner et déformer ou non la vidéo afin qu'elle s'adapte aux nouvelles dimensions. L'option Letterbox ajoute du noir de part et d'autre si les proportions ne sont pas préservées. Dans notre cas, elles le sont. Cette option n'aura donc aucun effet. Vous pouvez donc valider (voir Figure 7.13).

Figure 7.13

Compression
H-264.



La boîte de dialogue d'options affiche maintenant les informations en rapport avec la compression H-264 (voir Figure 7.14).

Figure 7.14

Compression
H-264.

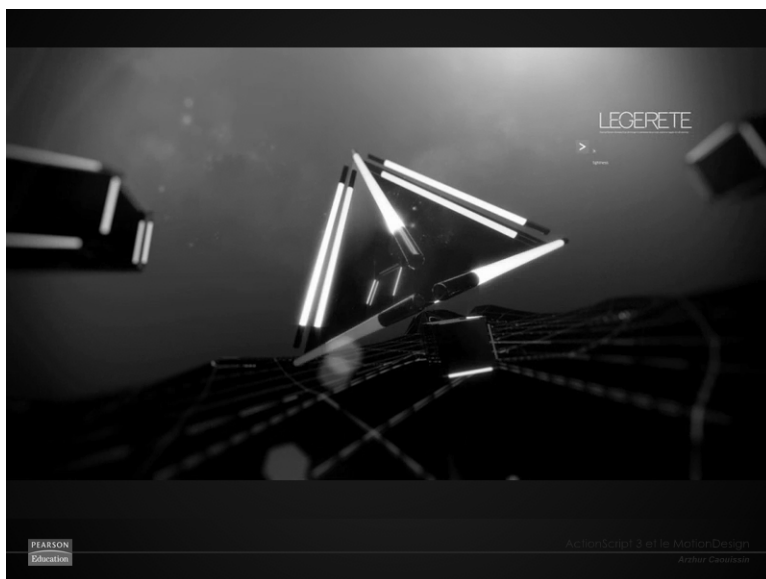


Attention, Quick Time conserve par défaut les précédents réglages lors de chaque nouvelle compression. Lorsque vous encoderez un nouveau document, pensez à initialiser le redimensionnement pour éviter d'affecter votre nouveau fichier.

Validez également la fenêtre d'options. Puis, renommez le fichier "gKaster-amusement-h264.mov" et confirmez l'enregistrement. L'encodage se termine. La vidéo peut être appelée directement depuis Flash, via un composant FLVPlayback (voir Figure 7.15).

Figure 7.15

Lecture d'une
vidéo Quick Time
H-264 dans un
document Flash.





Les formats pris en charge par le composant FLVPlayback (publié dans un document configuré pour ActionScript 3) sont : FLV, F4V, MP4, M4A, MOV, MP4V, 3GP et 3G2.

À retenir

- Quick Time permet d'exporter facilement une vidéo avec un encodage H-264 directement lisible par le composant FLVPlayback de Flash CS4.

Créer un lecteur vidéo personnalisé

Dans Flash, en utilisant le composant vidéo en vue d'intégrer une vidéo FLV ou F4V, une console de lecture est disponible par défaut. Elle permet de contrôler la vidéo sans programmation spécifique. Différents types de consoles sont disponibles à travers le paramètre Skin accessible dans l'Inspecteur de composants de chaque vidéo active.

Si vous utilisez une Skin prédéfinie, n'oubliez pas que Flash génère un fichier SWF pour chacune d'elle et qu'il ne faudra pas manquer de placer ce fichier sur le serveur d'hébergement, en même temps que votre document SWF de base et votre vidéo, sans quoi la console n'apparaîtrait pas.



Lorsque vous utilisez une Skin prédéfinie, Flash importe en réalité un fichier SWF qu'il génère à la volée, selon le type de Skin sélectionné dans la fenêtre Inspecteur de composants. Ces boutons sont standardisés. Mais vous pouvez modifier la forme intrinsèque des boutons déjà encapsulés dans ces Skins prédéfinies. Pour ce faire, dans le moteur de recherche de votre système, saisissez le nom de la Skin générée par Flash à la publication (par exemple `SkinOverA11`). Puis, repérez l'emplacement du fichier FLA, natif, ayant permis à Flash de générer ce fichier. Il suffit d'ouvrir ce FLA dans Flash, de le modifier et de publier un nouveau fichier en lieu et place du précédent. La console accessible depuis votre composant est instantanément mise à jour.

Pour personnaliser la forme des boutons, nous pouvons utiliser de simples symboles créés manuellement ou bien recourir à des composants séparés qui contrôlent la lecture de la vidéo. Ces composants sont également disponibles depuis la fenêtre des composants. L'avantage de ces éléments est qu'ils disposent déjà d'une structure animée (effet "roll-Over" intégré) et d'actions de contrôles préprogrammés.

Dans cette section, nous présentons uniquement ces boutons préprogrammés. Nous revenons sur les actions gérées manuellement en ActionScript, au Chapitre 8.



Exemples > `ch7_videoHD_3 fla`

Dans le document "`ch7_videoHD_3 fla`", sur la scène, un composant vidéo apparaît au-dessus des boutons de contrôle personnalisés (voir Figure 7.16).

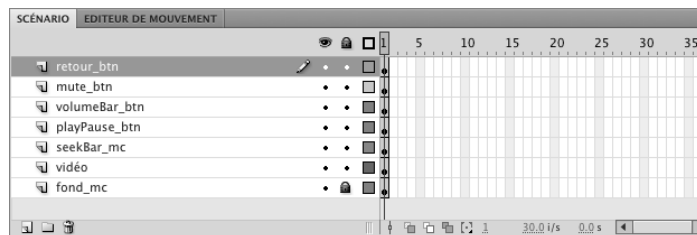
Dans le scénario, au-dessus du calque `fond_mc`, chaque symbole ou composant est réparti vers un calque distinct (voir Figure 7.17). Ils possèdent des noms d'occurrence, mais cela est facultatif puisqu'aucune action n'est associée à ces objets sinon celles déjà encapsulées dans les composants boutons eux-mêmes.

Figure 7.16

Composant
FLVPlayback
avec boutons
personnalisés.

**Figure 7.17**

Fenêtre
de scénario.

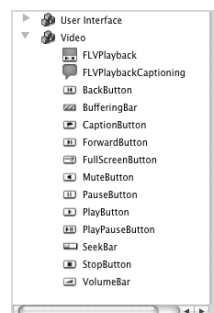


Pour insérer des boutons prédéfinis, prêts à utiliser, vous devez placer des éléments de la fenêtre Composants, sur la même scène que celle où figure le composant vidéo FLVPlayback. Flash, au moment de la publication, va automatiquement établir une liaison entre le composant vidéo et les boutons qui s'y réfèrent.

Pour placer des boutons préprogrammés sur la scène, affichez la fenêtre des composants via Fenêtre > Composants (voir Figure 7.18). Puis, dans la catégorie Vidéo, sélectionnez le bouton de votre choix. Puis, glissez-déposez les boutons individuellement sur la scène où se trouve le composant FLVPlayback.

Figure 7.18

Fenêtre
Composants.



Double-cliquez ensuite, dans la scène, sur chacun d’eux jusqu’à atteindre le graphisme modifiable pour éventuellement les personnaliser. Enregistrez et publiez. Les boutons insérés contrôlent la vidéo (voir Figure 7.19).

Figure 7.19

Aperçu du document après publication.



Vidéo

Définition des composants associés à FLVPlayback

Voici les commandes disponibles sous la forme de boutons prédéfinis, depuis la fenêtre Composants et leur définition :

- **BackButton.** Bouton qui permet théoriquement de rembobiner le flux vidéo. En réalité, il renvoie à la première image du flux vidéo (en progressif).
- **BufferingBar.** Barre de progression de mise en cache de la vidéo. Elle avance à mesure que la vidéo est chargée dans le cache du navigateur et signale à l'utilisateur la possibilité de naviguer à l'intérieur du flux déjà mis en cache.
- **CaptionButton.** Utilisé avec le composant FLVPlayback Captionning pour réaliser des sous-titrages (voir Chapitre 8).
- **ForwardButton.** Bouton d'accélération de la lecture de la vidéo. Attention, en flux progressif, seule la partie déjà chargée peut être atteinte par l'utilisateur.
- **FullScreenButton.** Active l'affichage en mode plein écran du contenu vidéo (voir aussi le Chapitre 15 pour la gestion du mode d'affichage en plein écran).
- **MuteButton.** Permet de stopper l'audio et de le réactiver, sur le même bouton.
- **PauseButton.** Arrête la lecture du flux vidéo, sans interrompre la connexion au serveur. La mise en cache se poursuit de manière transparente.
- **PlayButton.** Lit ou reprend la lecture d'un flux vidéo.
- **PlayPauseButton.** Propose sur le même bouton, une fonctionnalité de lecture de la vidéo lorsque celle-ci est en pause, et de mise en pause lorsqu'elle est en lecture.

- **SeekBar.** Barre de progression de la lecture de la vidéo. À la différence du composant **BufferingBar**, la barre affiche la position courante de la vidéo dans la scène. L'utilisateur peut déplacer la tête de lecture pour atteindre un autre extrait de la bande vidéo, dans la mesure, toujours, où le contenu appelé en flux progressif est déjà chargé par le navigateur.
- **StopButton.** Arrête la lecture vidéo et interrompt aussi son chargement. Cela libère le canal et les ressources graphiques sollicitées pendant la lecture de la vidéo.
- **VolumeBar.** Modifie le volume sonore de l'audio inclus dans la vidéo.



Il se peut que vous cherchiez à intégrer une vidéo dans un fichier SWF, lui-même importé dans un autre document SWF. Le fait de supprimer le SWF chargé dans la liste d'affichage (avec `removeChild`) ne permet pas, dans ce cas, d'interrompre la lecture de la vidéo. Pour ce faire, il convient d'associer, à la vidéo, des contrôles supplémentaires en ActionScript. Nous détaillons ces contrôles au chapitre suivant.

À retenir

- Le composant **FLVPlayback** met à disposition des consoles de lecture de la vidéo, personnalisables si l'on édite le FLA utilisé pour les générer, à partir du document source disponible dans le dossier de l'application Flash.
- Des boutons composants sont utilisables individuellement et peuvent être personnalisés directement depuis l'interface auteur, dans le scénario. Il ne requièrent en outre aucune programmation spécifique pour fonctionner.

Créer un lecteur vidéo H-264 pour Flash 6 et plus

Si l'utilisation du composant offre une plus grande souplesse de manipulation pour des formats standards comme le FLV et le F4V, il ne permet pas de lire des fichiers vidéo codés en H-264 avec des versions de documents antérieures à la version 10 (CS4), car le codec H-264 n'est implémenté dans le composant **FLVPlayback** que depuis cette version.

Pour permettre de lire un flux HD (H-264) dans des versions de Flash antérieures à Flash 10, nous créons un lecteur vidéo manuellement, en ActionScript, à l'aide de la classe **NetStream**. Cette classe étant apparue avec Flash 6, nous pouvons programmer l'affichage de la vidéo en haute définition à partir de Flash 6 et pour les versions ultérieures.

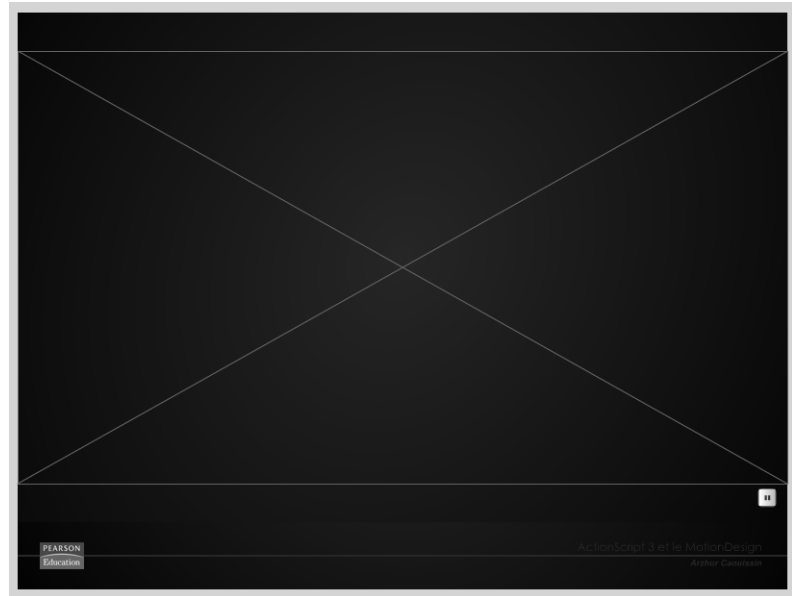


Exemples > `ch7_videoHD_4 fla`

Dans le document "`ch7_videoHD_4 fla`", sur la scène, apparaît une vidéo vide et transparente. C'est un objet vidéo importé de la bibliothèque et utilisé avec la classe **NetStream** (voir Figure 7.20). À droite et en bas, deux boutons `lire_btn` et `pause_btn` sont superposés.

Figure 7.20

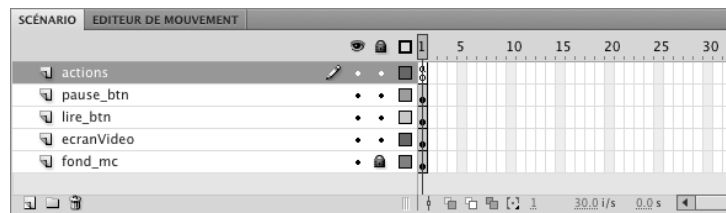
Aperçu de la scène principale.



Dans la fenêtre de scénario, au-dessus du calque `fond_mc`, un calque affiche l'objet vidéo. Deux calques distribuent respectivement les deux boutons `lire_btn` et `pause_btn`. Un autre calque affiche les actions (voir Figure 7.21).

Figure 7.21

Aperçu du scénario de la scène principale.



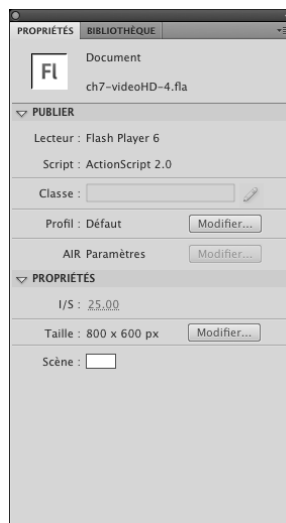
Les propriétés du document affichent enfin un format d'exportation Flash 6 et ActionScript en version 2 (voir Figure 7.22).

Les objets vidéo sont nativement disponibles dans tous les documents Flash, depuis la bibliothèque. L'objet qui figure dans ce document est donc extrait de la bibliothèque. Pour extraire un objet vidéo de la bibliothèque, procédez comme suit :

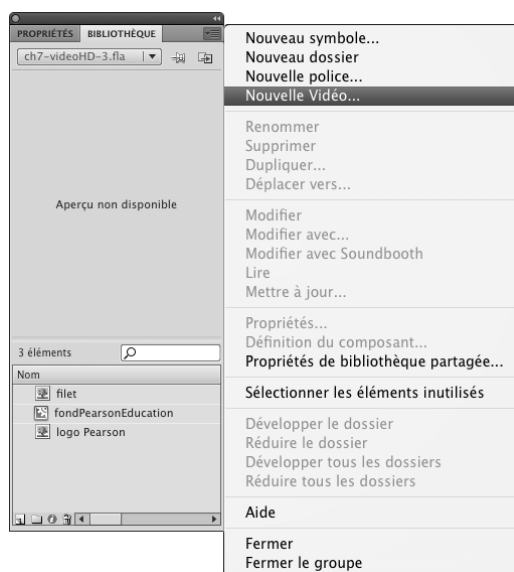
1. Affichez la bibliothèque (Fenêtre > Bibliothèque ou Cmd+L sur Mac ou Ctrl+L sur Windows).
2. Pour insérer un objet vidéo, supprimez éventuellement l'objet déjà en place sur la scène et dans la bibliothèque.
3. Dans le menu contextuel de la bibliothèque, sélectionnez l'option Nouvelle Vidéo (voir Figure 7.23).
4. Dans la boîte de dialogue, attribuez un nom d'objet, par exemple `ecranVideo` (voir Figure 7.24). Puis validez.

Figure 7.22

Aperçu des propriétés de la scène principale.

**Figure 7.23**

Création d'un nouvel objet Vidéo.

**Figure 7.24**

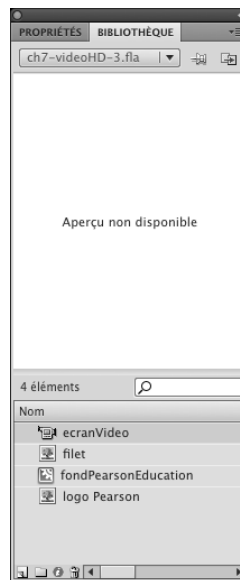
Propriétés de la vidéo.



5. Le nouvel objet apparaît dans la bibliothèque (voir Figure 7.25).

Figure 7.25

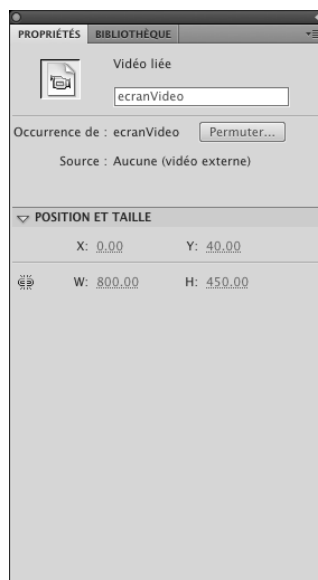
Fenêtre
Bibliothèque.



6. Glisser-déposez cet objet sur la scène et, depuis l'Inspecteur de propriétés, donnez-lui un nom d'occurrence, par exemple : `ecranVideo`. Puis, redimensionnez-le à la taille du flux vidéo à importer, par exemple 800×450 pixels (voir Figure 7.26).

Figure 7.26

Propriétés de
l'occurrence de
l'objet vidéo placé
sur la scène.



Dans la fenêtre Actions, nous pouvons lire le code suivant, en ActionScript 2 :

```
// Lecteur vidéo Flash 6
connexion = new NetConnection();
connexion.connect(null);
chargementContinu = new NetStream(connexion);
ecranVideo.attachVideo(chargementContinu);
chargementContinu.play("gKaster/gKaster-amusement.f4v");

// gestion du bouton Play/Pause

lire_btn._visible=false;
pause_btn._visible=true;

lire_btn.onPress=function () {
    chargementContinu.pause();
    lire_btn._visible=false;
    pause_btn._visible=true;
}
pause_btn.onPress=function () {
    chargementContinu.pause();
    lire_btn._visible=true;
    pause_btn._visible=false;
}
```

Ce code est structuré en deux parties. La première gère l’affichage de la vidéo. La seconde organise les boutons `lire_btn` et `pause_btn` qui contrôlent la vidéo.

Dans la première partie, nous spécifions d’abord, sans typage spécifique (AS2 autorise), une variable intitulée `connexion` qui désigne l’activation d’un flux Internet distant :

```
connexion = new NetConnection();
```

Plus loin, nous initialisons cette connexion :

```
connexion.connect(null);
```

La valeur `null`, renseignée en paramètre, spécifie que l’on se connecte en relatif sur le même emplacement que la source SWF. Sinon il serait également possible d’user de `http://` ou de `rtmp://` pour désigner une référence absolue.

Puis, nous créons une deuxième variable qui active le transfert d’un flux à chargement progressif (faux *streaming*). Nous précisons, en paramètre de la méthode `NetStream` que ce flux concerne la connexion préalablement définie :

```
chargementContinu = new NetStream(connexion);
```

Nous attachons ensuite ce flux progressif à l’occurrence vidéo qui figure sur la scène (`ecranVideo`) :

```
ecranVideo.attachVideo(chargementContinu);
```

Enfin, nous spécifions que ce flux doit lire le fichier localisé dans le chemin défini entre les guillemets :

```
chargementContinu.play("gKaster/gKaster-amusement.f4v");
```

Même si le format F4V, comme nous l’avons précisé, n’est pas reconnu dans les anciennes versions de Flash, ce qui est appelé ici est bien un fichier encodé en H-264. Flash, en réalité,

fait abstraction de l'extension lors de la lecture de vidéos. C'est pourquoi nous pouvons appeler directement le fichier nommé F4V. Le composant FLVPlayback des anciennes versions ne sait pas interpréter le F4V, mais la classe NetStream, elle, le fait.

Dans la seconde partie du code, deux actions contrôlent chaque bouton individuellement. Le premier, lire_btn, se masque lui-même et affiche le bouton pause_btn. Simultanément, il reprend la lecture là où elle était arrêtée grâce à la méthode `pause()` :

```
lire_btn.onPress=function () {  
    chargementContinu.pause();  
    lire_btn._visible=false;  
    pause_btn._visible=true;  
}
```

De même, le bouton pause_btn se fait disparaître et réaffiche le bouton lire_btn tout en interrompant la lecture en cours :

```
pause_btn.onPress=function () {  
    chargementContinu.pause();  
    lire_btn._visible=true;  
    pause_btn._visible=false;  
}
```

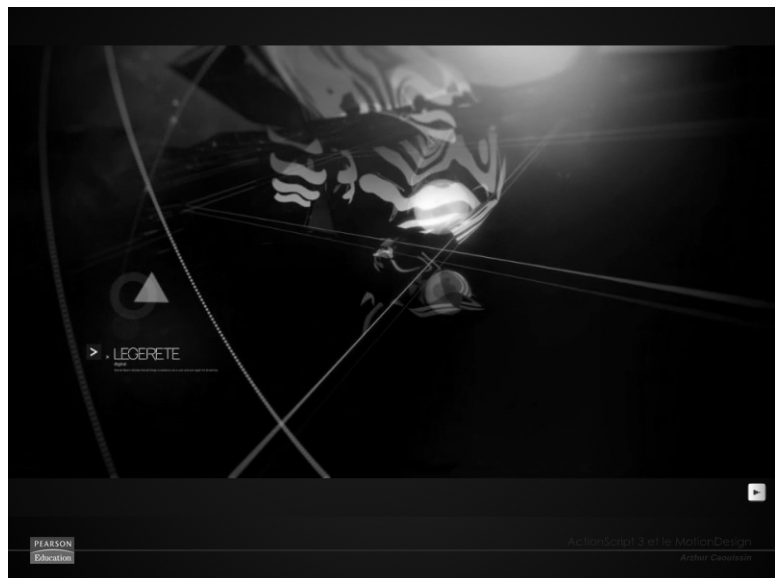
À l'initialisation, puisque la vidéo est lue dès le chargement, juste avant les actions attachées aux deux boutons, nous spécifions que le bouton pause_btn est visible et que lire_btn est masqué :

```
lire_btn._visible=false;  
pause_btn._visible=true;
```

En publiant le document, la vidéo F4V joue instantanément en haute définition dans un document Flash 6 codé en ActionScript 2. Les boutons lire et pause contrôlent la vidéo (voir Figure 7.27).

Figure 7.27

Lecture d'une vidéo Quick Time H-264 dans un document Flash.



À retenir

- Il est possible de publier une vidéo en haute définition pour un lecteur Flash 6 ou de version ultérieure. Pour cela, nous utilisons la classe NetStream.
- Pour contrôler la lecture du flux vidéo HD avec NetStream, nous codons et exportons en ActionScript 2.

Agrandir une vidéo sans perte

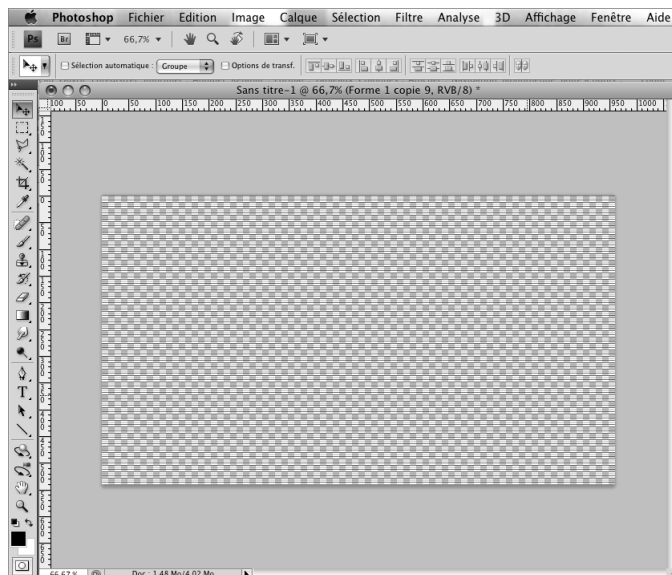
La difficulté de la vidéo pour le Web réside dans le bon compromis entre une compression avec un contenu accessible et une qualité qui, elle, valorise le contenu au détriment de l'accessibilité. Avec des vidéos traitées en haute définition, le compromis devient presque impossible à résoudre dans des contextes où l'image est très riche (présence de bruit continu, images toutes différentes, audio avec une belle amplitude, animation de durée conséquente, etc). Pour permettre une compression forte tout en conservant l'illusion d'une image de qualité, il existe une astuce. Comme dans l'édition papier, et dans la presse quotidienne en particulier, pour compenser la perte engendrée par une image allégée, nous lui appliquons une trame.

Vous pouvez appliquer une trame dans Flash au moyen d'une image tramée et transparente placée dans un MovieClip auquel vous appliquez une propriété d'affichage de type Incrustation ou Produit. Mais, cela sollicite de manière importante la carte vidéo de l'utilisateur. Nous préférons alors intégrer cette trame directement dans le flux vidéo :

1. Dans un logiciel graphique, comme Photoshop, dessinez cette trame en reproduisant sur une image de dimensions identiques à celles de la vidéo, un motif (un filet, un damier, des points, des croix ou un effet trame de demi-teinte, par exemple) – voir Figure 7.28.

Figure 7.28

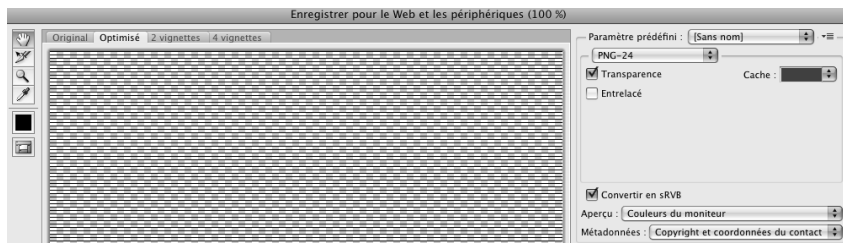
Création dans Photoshop d'une trame de type filaire.



2. Puis, exportez cette image en PNG-24, afin d'en préserver la transparence, en faisant Fichier > Exporter pour le Web et les périphériques.
3. Dans la boîte de dialogue d'enregistrement, à droite, sélectionnez l'option PNG-24 et spécifiez bien une transparence en cochant cette option (voir Figure 7.29).

Figure 7.29

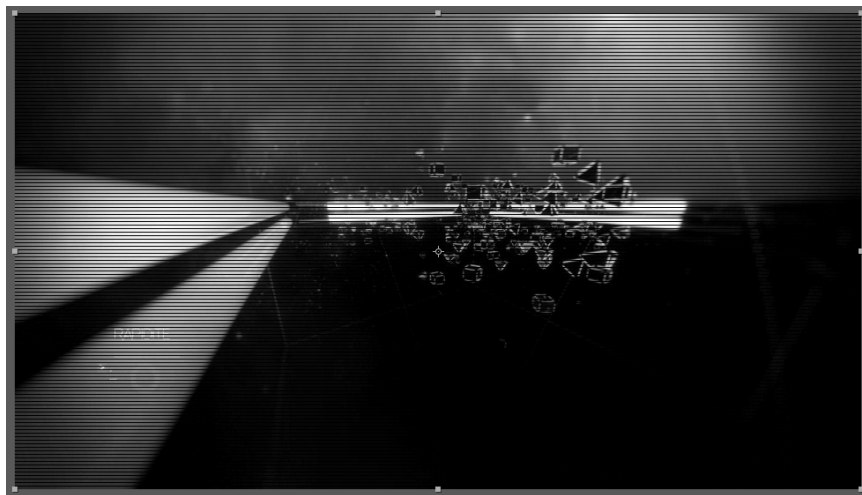
Exporter au format PNG-24, avec la transparence.



4. Dans After Effects par exemple, importez l'image en faisant Ctrl+I (Windows) ou Cmd+I (Mac).
5. Puis, glissez l'image sur la vidéo déjà en place dans la scène (voir Figure 7.30). Au besoin, ajustez le paramètre Mode disponible sur le calque dans le scénario, en choisissant un mode de type Produit, Incrustation ou autre.

Figure 7.30

Appliquer la trame sur la vidéo, dans After Effects.



6. Puis, encodez la vidéo.
- Pour le logiciel Motion, procédez de même.
1. Depuis la fenêtre Bibliothèque, glissez-déposez l'image réalisée dans Photoshop sur votre scène.
 2. Puis, exportez la vidéo au format Quick Time.

À retenir

- Pour agrandir une vidéo, nous conservons une compression forte, mais compensons l'affichage par l'ajout d'un tramage.
- Il est préférable d'intégrer ce tramage dans le flux du signal vidéo plutôt que dans le document Flash afin d'optimiser les ressources graphiques de l'utilisateur.

Synthèse

Dans ce chapitre, vous avez appris à gérer l'affichage de flux vidéo en haute définition, codés en H-264, avec Adobe Media Encoder ou tout autre type de source et notamment avec les solutions Apple. Vous avez également appris à intégrer une vidéo HD pour les anciennes versions de Flash, jusqu'à la version 6. Vous avez appris à personnaliser les boutons de contrôle de lecture d'une vidéo et à contourner les contraintes de poids liées à une forte compression grâce à des astuces inspirées de l'univers de la presse. Vous êtes à mesure de réaliser désormais des sites contenant des vidéos de qualité professionnelle et haut de gamme, compatibles avec toutes les configurations d'utilisateur.

8

La vidéo interactive

Introduction

Tout l'intérêt de l'intégration de la vidéo dans Flash, qu'elle soit standard, composite ou en haute définition, est qu'en plus d'offrir un espace confortable pour son affichage, vous pouvez y associer des comportements. Dans ce chapitre, nous allons voir comment interagir avec la vidéo aussi bien à travers la navigation (sur actions de l'utilisateur) qu'à travers des événements (sur des actions exécutées par la vidéo durant son déroulement). En plus de toute cette interactivité, nous allons aborder la manière de réaliser une boucle et comment, dans le contexte de contenus SWF imbriqués, interrompre réellement le flux vidéo. À l'issue du chapitre, vous serez en mesure de créer des interfaces vidéo interactives riches, en ligne, mais aussi pour des systèmes hors ligne.

Dans ce chapitre, nous utilisons la bande démo de la société gKaster, généreusement mise à disposition pour cette présentation (www.gKaster.com).

Contrôles de base de la vidéo

Les premières actions nécessaires pour la gestion d'une vidéo sont les contrôles de lecture. Dans cette première section, nous abordons les actions qui permettent de lire, arrêter, accélérer et rembobiner un flux vidéo, mais aussi modifier le volume sonore d'une vidéo associée à un composant FLVPlayback.



Exemples > ch8_videoInteractive_1.fla

Dans le document "ch8_videoInteractive_1.fla", sur la scène, un composant vidéo joue la bande démo de la société gKaster. Au-dessous, une console contient différents boutons qui contrôlent le flux vidéo (voir Figure 8.1). Dans cette console, chaque symbole est disposé sur un calque séparé et possède un nom d'occurrence lui permettant de recevoir une action. Le symbole `audio_mc`, lui, contient deux `MovieClip` qui affichent chacun un état activé ou non activé, pour le contrôle du son. Ils possèdent également leur propre nom d'occurrence.

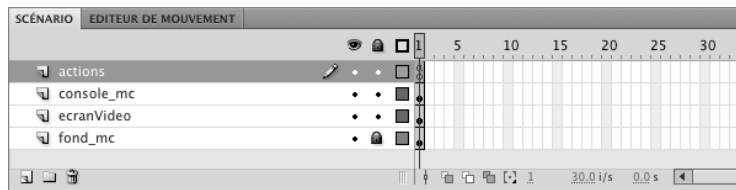
Dans la fenêtre de scénario de la scène principale, au-dessus du calque `fond_mc`, apparaissent la console, le composant et un calque actions (voir Figure 8.2).

Figure 8.1

Aperçu après publication.

**Figure 8.2**

Aperçu du scénario de la scène principale.



Dans le calque actions, nous pouvons lire le code suivant :

```
// lecture automatique
ecranVideo.autoPlay=true;

// mise en cache
ecranVideo.bufferTime=3000;

// lire
console_mc.lire_btn.addEventListener(MouseEvent.CLICK,jouerVideo);
function jouerVideo (evt:MouseEvent) {
    écranVideo.play();
}

// arrêter
console_mc.stop_btn.addEventListener(MouseEvent.CLICK,stopperVideo);
function stopperVideo (evt:MouseEvent) {
    écranVideo.stop();
}

// suite
console_mc.suite_btn.addEventListener(MouseEvent.CLICK,suiteVideo);
function suiteVideo (evt:MouseEvent) {
    écranVideo.seek(ecranVideo.playheadTime+2)
}
```

```
// retour
console_mc.retour_btn.addEventListener(MouseEvent.CLICK,retourVideo);
function retourVideo (evt:MouseEvent) {
    ecranVideo.seek(ecranVideo.playheadTime-2)
}

// Audio
console_mc.audio_mc.audioOff_mc.visible=false;
console_mc.audio_mc.audioOff_mc.buttonMode=true;

console_mc.audio_mc.addEventListener(MouseEvent.CLICK,audioVideo);
function audioVideo (evt:MouseEvent) {
    if (ecranVideo.volume>0) {
        ecranVideo.volume=0;
        evt.target.visible=false;
        console_mc.audio_mc.audioOff_mc.visible=true;
    } else {
        ecranVideo.volume=1;
        evt.target.visible=false;
        console_mc.audio_mc.audioOn_mc.visible=true;
    }
}
```

Voici le descriptif détaillé des différentes fonctionnalités rassemblées dans cette console.

Lecture automatique

La première ligne spécifie si le composant nommé `ecranVideo` doit jouer automatiquement dès l'affichage de l'animation (`true`) où s'il doit attendre une action de l'utilisateur (`false`). L'action est activée sur `true`. La vidéo joue donc automatiquement dès la publication :

```
// lecture automatique
ecranVideo.autoPlay=true;
```

Mise en cache

Une seconde action affecte directement le signal vidéo contrôlé par le composant vidéo. C'est la mise en cache. La propriété `bufferTime` permet d'indiquer le nombre de millisecondes à attendre avant d'exécuter la lecture du flux vidéo. Cette indication permet de réduire les risques de rupture de flux pour les connexions faibles. Ici, la valeur portée à 3 000 désigne une attente de trois secondes avant le démarrage de la vidéo :

```
// mise en cache
ecranVideo.bufferTime=3000;
```

La valeur assignée par défaut à la propriété `bufferTime` est 5 000. Elle ne peut pas être modifiée depuis l'Inspecteur de composants.

Lire la vidéo

Plus loin, un écouteur est attaché au symbole bouton `lire_btn` et indique de lire le flux vidéo avec l'action `play()` :

```
// lire
console_mc.lire_btn.addEventListener(MouseEvent.CLICK, jouerVideo);
function jouerVideo (evt:MouseEvent) {
    ecranVideo.play();
}
```

Arrêt de la vidéo

À la suite, un écouteur est attaché au symbole bouton `stop_btn` et indique de stopper la progression du flux avec l'action `stop()` :

```
// arrêter
console_mc.stop_btn.addEventListener(MouseEvent.CLICK, stopperVideo);
function stopperVideo (evt:MouseEvent) {
    ecranVideo.stop();
}
```

`stop` interrompt également la mise en cache. Pour arrêter et reprendre la lecture de la vidéo sans interrompre sa mise en cache, utilisez `pause()`.

Accélérer la vidéo

Il est possible d'accélérer la lecture de la vidéo en positionnant la tête de lecture de la vidéo à une image plus éloignée que l'image courante. Pour effectuer ce calcul, nous affectons d'abord la propriété `playheadTime` au composant vidéo pour permettre de capturer la position courante de la tête de lecture. Puis, nous augmentons la valeur, obtenue par cette propriété, de quelques images ou secondes. Nous passons ensuite directement cette valeur en paramètre de la méthode `seek()` qui permet de repositionner la tête de lecture dans la vidéo :

```
// suite
console_mc.suite_btn.addEventListener(MouseEvent.CLICK, suiteVideo);
function suiteVideo (evt:MouseEvent) {
    ecranVideo.seek(ecranVideo.playheadTime+2)
}
```

Les valeurs que nous passons en paramètre sont des nombres. La méthode `seek()` renvoie la tête de lecture à une position qui correspond toujours à l'image-clé suivante la plus proche dans le flux vidéo. La précision du ciblage dépend donc directement du nombre d'images-clés encodées dans le flux vidéo.

La vidéo que nous traitons a été encodée avec un nombre d'images-clés défini sur Automatique. Cette valeur, dans Adobe Media Encoder, spécifie en réalité un intervalle de deux secondes entre chaque image-clé. Pour obtenir une navigation plus précise que par pas de deux secondes, il convient donc de revenir sur la compression en modifiant cette valeur à un intervalle plus serré. Rappelez-vous cependant que plus un intervalle est serré, plus l'encodeur ajoute des images-clés. Or, chaque image-clé augmente le poids et, à poids égal, à taux

de compression égal, à débit égal, l'ajout d'images-clés réduit la place disponible pour encoder les autres images. Cela détériore donc l'ensemble de la vidéo. Un paradoxe que l'on peut résoudre très simplement : en augmentant les valeurs des réglages de compression et de débit.

Le pas d'incrémentation utilisé dans notre exemple est 2 (deux secondes). La tête de lecture est donc déplacée à l'image-clé la plus proche à deux secondes d'intervalle après la position courante du flux vidéo et reprend la lecture de la vidéo à partir de cette image.

Dans notre exemple, le repositionnement appelle donc toujours la deuxième image-clé située après la position courante de la tête de lecture et poursuit la lecture de la vidéo à partir de cette image, car c'est toujours l'image-clé suivante, la plus proche de l'image invoquée, qui est affichée.

Notez qu'une valeur de recherche élevée et une vidéo avec des plans assez longs peuvent contribuer à masquer le manque de précision induit par la méthode `seek()` et par un type de compression impliquant un faible nombre d'images-clés.

Rembobiner la vidéo

De la même manière que nous pouvons accélérer la vidéo, nous la rebobinons avec la méthode `seek()` et la propriété `playheadTime`. Mais, ici, nous retranchons la valeur à la position courante du flux vidéo de sorte à revenir deux seconde en arrière, à chaque clic sur le bouton retour :

```
// retour
console_mc.retour_btn.addEventListener(MouseEvent.CLICK,retourVideo);
function retourVideo (evt:MouseEvent) {
    ecranVideo.seek(ecranVideo.playheadTime-2)
}
```

Modifier le volume audio

La méthode `volume()` permet de modifier le volume sonore global de la vidéo. La valeur à passer en paramètre est 1 pour un volume normal et 0 pour un son muet. Les valeurs décimales intermédiaires permettent de nuancer le volume.

Dans notre exemple, dans le MovieClip `console_mc`, nous pouvons identifier le symbole `audio_mc`. Ce dernier contient lui-même deux autres clips dont `audioOff_mc` qui affiche un trait rouge et `audioOn_mc` qui reste neutre.

Nous spécifions ici qu'en cliquant sur chaque symbole contenu dans `audio_mc` (`evt.target`), nous modifions le volume audio de la vidéo. Au premier clic, le son devient muet, et au suivant, il redevient normal, et ainsi de suite. Une condition permet de vérifier si le volume est déjà muet ou non et inverse la valeur selon le résultat.

Pour que le dispositif soit plus ergonomique, nous ajoutons un contrôle de visibilité sur chacun des boutons de sorte que l'un disparaît à chaque fois qu'on l'active, et laisse alors l'autre prendre sa place :

```
// Audio
console_mc.audio_mc.audioOff_mc.visible=false;
console_mc.audio_mc.audioOff_mc.buttonMode=true;

console_mc.audio_mc.addEventListener(MouseEvent.CLICK,audioVideo);
function audioVideo (evt:MouseEvent) {
    if (ecranVideo.volume>0) {
        ekranVideo.volume=0;
        evt.target.visible=false;
        console_mc.audio_mc.audioOff_mc.visible=true;
    } else {
        ekranVideo.volume=1;
        evt.target.visible=false;
        console_mc.audio_mc.audioOn_mc.visible=true;
    }
}
```

Augmenter et diminuer progressivement le son



Exemples > ch8_videoInteractive_2 fla

Nous avons vu qu'il est possible de modifier une valeur en l'incrémentant au sein d'un gestionnaire de type `Event.ENTER_FRAME` (voir Chapitre 1). Vous pouvez donc aussi modifier le son, sur action utilisateur, en appelant un écouteur qui active la modification de l'audio tant que celui-ci n'atteint pas une certaine valeur. Nous utilisons pour ce faire un gestionnaire de type `Event.ENTER_FRAME` et des structures conditionnelles. Nous obtenons ceci :

```
// Audio
console_mc.audio_mc.audioOff_mc.visible=false;
console_mc.audio_mc.audioOff_mc.buttonMode=true;

console_mc.audio_mc.audioOn_mc.addEventListener(MouseEvent.CLICK,fonctionBaisser);
function fonctionBaisser (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME,reduceAudio);
}
//
function reduceAudio (evt:Event) {
    if (ecranVideo.volume>0) {
        ekranVideo.volume-=0.01;
        trace(ecranVideo.volume)
        if (ecranVideo.volume<=0.01) {
            ekranVideo.volume=0;
            removeEventListener(Event.ENTER_FRAME,reduceAudio);
            console_mc.audio_mc.audioOn_mc.visible=false;
            console_mc.audio_mc.audioOff_mc.visible=true;
        }
    }
}
```

```
console_mc.audio_mc.audioOff_mc.addEventListener(MouseEvent.CLICK, fonctionMonter);
function fonctionMonter (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME, monterAudio);
}
//
function monterAudio (evt:Event) {
    if (ecranVideo.volume<1) {
        ekranVideo.volume+=0.01;
        trace(ekranVideo.volume)
        if (ekranVideo.volume>=0.99) {
            ekranVideo.volume=1;
            removeEventListener(Event.ENTER_FRAME, monterAudio);
            console_mc.audio_mc.audioOn_mc.visible=true;
            console_mc.audio_mc.audioOff_mc.visible=false;
        }
    }
}
```

Dans cet exemple, chaque bouton exécute une fonction qui lui est propre. Dans cette fonction, le gestionnaire `Event.ENTER_FRAME` appelle une autre fonction. C'est alors que la valeur de l'audio est, soit augmentée, soit diminuée, selon la fonction qui est exécutée (suivant le bouton qui est cliqué). Une fois que la valeur est intégralement renversée, alors, une instruction interrompt la fonction.

Pour modifier la vitesse de progression du volume, il suffit de modifier la valeur du pas d'incréméntation, ici spécifiée à `0.01`.

À retenir

- Il est possible de personnaliser une console de contrôle vidéo en associant à des clips des fonctions qui affectent les propriétés du composant vidéo en cours d'exécution.
- Il est possible d'accélérer ou de rembobiner une vidéo en utilisant la propriété `playheadTime` et `seek()`. Mais cette technique offre une précision relative au nombre d'images-clés encodées dans le flux vidéo.
- Il est possible de modifier le son d'un flux audio en contrôlant la propriété `volume`. Une variation progressive de l'audio peut être effectuée grâce à un gestionnaire de type `Event.ENTER_FRAME`.

Chapitrage vidéo

La navigation au sein d'une vidéo est très simple à mettre en place. Il suffit d'utiliser la méthode `seek()` que nous venons déjà de rencontrer. Nous spécifions alors, en paramètre de cette méthode, le `timecode` à atteindre. Le `timecode` représente la position des images d'une vidéo et s'exprime en secondes. Un `timecode` de 12 désigne à la tête de lecture d'atteinte l'image située à la seconde 12 de la vidéo.

Naturellement, comme évoqué plus haut, nous devons aussi considérer que le flux vidéo dispose d'un nombre confortable d'images-clés ou alors, nous acceptons que le ciblage fluctue plus ou moins à une ou deux secondes près.

Pour définir un timecode précisément, considérons l'exemple d'une vidéo d'une cadence de 25 ips pour laquelle une image vaut 4 centièmes de secondes (100 centièmes de seconde / 25 images = 0,04 seconde). Pour cette vidéo, l'expression suivante appelle la soixante-septième seconde :

```
ecranVideo.seek(67) ;
```

Mais, l'expression suivante appelle la troisième image de la soixante-septième seconde (soit 67 secondes + $3 \times 0,04$). Si l'image appelée n'est pas une image-clé, comme vu précédemment, c'est l'image-clé suivante la plus proche qui est affichée et c'est à partir de cette image-clé que va se prolonger la vidéo :

```
ecranVideo.seek(67.12) ;
```

Dans l'exemple suivant, nous utilisons la méthode `seek()` à travers une série de vignettes afin de créer un système de chapitrage. Mais, nous allons plus loin que dans la section précédente en optimisant ici le code et en rassemblant d'abord toutes les conditions dans une seule et même fonction. À l'intérieur de cette fonction, nous ajoutons aussi une propriété qui permet de charger éventuellement une autre vidéo, en lieu et place de la vidéo active. Nous combinons donc deux méthodes : `seek()` et `source()`.

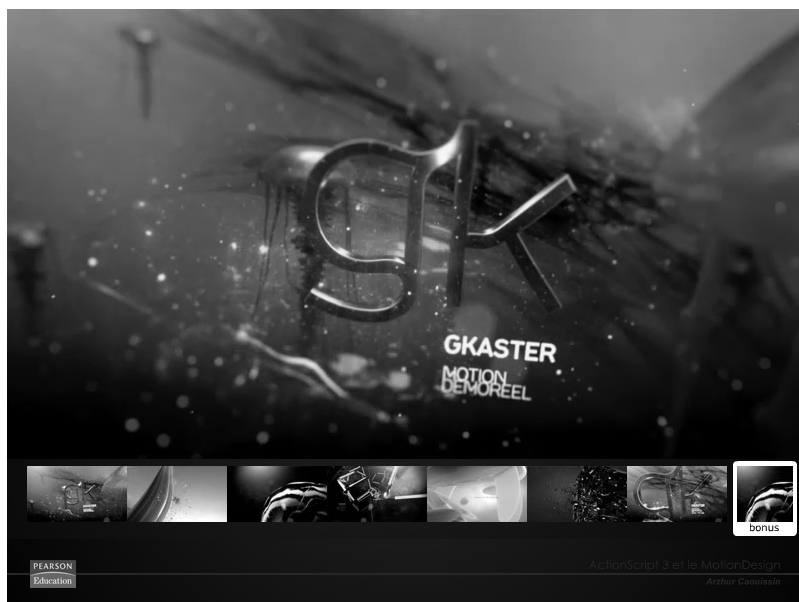


Exemples > ch8_videoInteractive_3 fla

Dans le document "ch8_videoInteractive_3 fla", sur la scène principale, se trouve un composant qui exécute directement la vidéo à la publication du document. En dessous, un menu est composé de plusieurs vignettes. Chacune de ces vignettes est isolée dans un MovieClip et possède un nom d'occurrence (voir Figure 8.3).

Figure 8.3

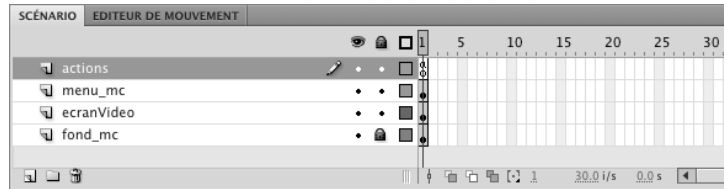
Aperçu du document après publication.



Dans le scénario, au-dessus de calque `fond_mc`, nous identifions le composant vidéo, le menu et un calque actions (voir Figure 8.4).

Figure 8.4

Aperçu du scénario de la scène principale.



Dans le calque actions, nous lisons le code suivant :

```
var cheminVideo:String;
var timeCode:Number=0;

menu_mc.addEventListener(MouseEvent.CLICK,timeCode1);
function timeCode1 (evt:MouseEvent) {
    if (evt.target.name=="lien1_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=0.2;
    }
    if (evt.target.name=="lien2_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=10;
    }
    if (evt.target.name=="lien3_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=19;
    }
    if (evt.target.name=="lien4_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=45;
    }
    if (evt.target.name=="lien5_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=66;
    }
    if (evt.target.name=="lien6_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=92;
    }
    if (evt.target.name=="lien7_btn") {
        cheminVideo="gKaster/gKaster-demoreel.f4v";
        timeCode=119.04;
    }
    if (evt.target.name=="bonus_btn") {
        cheminVideo="gKaster/gKaster-amusement.f4v";
        timeCode=0;
    }
    ecranVideo.source=cheminVideo;
    ecranVideo.seek(timeCode);
}
```

Nous déclarons en premier lieu deux variables :

```
var cheminVideo:String;  
var timeCode:Number=0;
```

La première désigne une chaîne de caractères qui véhiculera le chemin d'un fichier vidéo, disponible dans notre projet. Chaque lien cliqué va ainsi pouvoir renseigner cette valeur pour cibler le fichier qui lui est propre. Plus loin dans le code, une instruction reprend la valeur alors renseignée pour appeler le fichier correspondant avec la méthode `source()`.

La deuxième variable est un nombre et initialise les timecodes avant qu'ils ne soient éventuellement modifiés à travers la fonction.

Ensuite, un écouteur est attaché au menu et non aux vignettes elles-mêmes. Cela permet d'introduire des conditions qui définissent, selon l'objet du menu qui est cliqué (`evt.target.name`), telle ou telle instruction (voir Chapitre 5 pour en savoir plus sur la propriété `target`) :

```
if (evt.target.name=="lien1_btn") {  
    cheminVideo="gKaster/gKaster-demoreel.f4v";  
    timeCode=0.2;  
}
```

La condition vérifie que le nom de l'objet cliqué correspond bien à celui spécifié entre parenthèses. Lorsque la valeur est vérifiée, la variable `cheminVideo` est renseignée, ainsi que le `timecode`. Une condition est créée pour chaque bouton.

En fin de programme, les deux variables sont utilisées pour activer le chapitrage :

```
ecranVideo.source=cheminVideo;  
ecranVideo.seek(timeCode);
```

Vous remarquez le dernier bouton bonus, qui se distingue des autres en cela qu'il appelle un fichier différent. Notez que nous exécutons le programme localement, et donc, que les vidéos sont chargées instantanément. Nous pourrions donc spécifier un `timecode` pour cette nouvelle vidéo. Mais, n'oubliez pas que l'on ne peut atteindre une image d'un fichier vidéo que si l'image appelée est déjà chargée. Nous ne recommandons donc pas de spécifier une autre valeur que 0 lorsqu'une nouvelle vidéo est appelée. Le seul moyen de permettre d'atteindre directement une image d'une vidéo non chargée est d'utiliser la technologie Flash Media Server, qui autorise la diffusion en vrai *streaming* (en continu).

Nous remarquons ici que le chapitrage peut appeler indifféremment des séquences dans un même flux vidéo (avec `seek`) que plusieurs fichiers vidéo distincts (avec `source`). Notez que la création de flux séparés (avec `source`) offre une plus grande souplesse dans la navigation, car l'ensemble de la vidéo n'a alors pas besoin d'être chargée pour permettre d'accéder à d'autres chapitres. Les références appelées à chaque requête (avec `source`) sont toujours de nouveaux flux vidéo, indépendants, qui se substituent à la vidéo en cours de lecture. Que les chapitres soient constitués de vidéos distinctes ou contenues dans une seule signal vidéo, nous utilisons un seul, même et unique, composant.



Adobe propose une technologie serveur adaptée à la gestion de flux vidéo haute définition en continu. Cette technologie se nomme Flash Media Server. Vous trouverez des informations sur son utilisation ainsi qu'un serveur de test d'hébergement à l'adresse suivante : http://www.streame-dia.eu/#news_fr_5.html.

Chapitrage vidéo avec les points de repère

Il est également possible de créer un système de chapitrage à partir de points de repère. Nous utilisons alors la méthode `seekToNavCuePoint()`. Cette technique est plus précise car il est possible de contrôler la position des images-clés. Reportez-vous à la section "Synchroniser des actions avec les points de repère" pour en savoir plus sur cette méthode.

À retenir

- Pour créer un système de chapitrage simple, nous utilisons la méthode `seek()`. Mais le nombre d'images-clés encodées dans le flux vidéo détermine la précision du ciblage.
- Le chapitrage peut appeler aussi bien des séquences dans un même flux vidéo que plusieurs fichiers vidéo distincts. Mais, étant donné que le ciblage ne permet d'atteindre que les flux déjà chargés ou qui démarrent, l'option avec des fichiers séparés demeure la plus confortable pour l'utilisateur.

Sous-titrage vidéo

Flash met à disposition un composant qui simplifie la gestion des sous-titres de la vidéo. Vous devez pour cela utiliser d'abord un composant `FLVPlayback` pour y charger une vidéo, puis, créer un fichier XML qui contient le texte pour les sous-titres en respectant un format bien défini. Enfin, vous devez ajouter sur la scène un composant `FLVPlaybackCaptioning` qui se charge de placer le texte du fichier XML dans le champ de texte de votre choix.

Dans cet exemple, nous ajoutons des sous-titres qui accompagnent une création autour d'un poème de Dan Andersson – écrivain suédois – (voir Figure 8.5). Les textes sont stockés dans un fichier XML.

Figure 8.5

Aperçu du document après publication.





Exemples > ch8_videoInteractive_4.fla

Dans le document "ch8_videoInteractive_4.fla", la scène principale affiche un composant vidéo, un texte dynamique déjà formaté avec une typo embarquée (voir Chapitre 15 pour en savoir plus sur la typographie) et un composant FLVPlaybackCaptioning situé hors champ (voir Figure 8.6). Chaque objet possède un nom d'occurrence. La vidéo appelée se nomme "gkaster-c19.f4v".

Figure 8.6

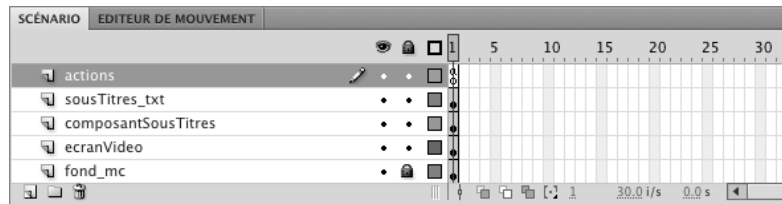
Aperçu de la scène principale.



Dans la fenêtre de scénario, au-dessus du calque fond_mc, les trois objets sont clairement répartis vers des calques distincts (voir Figure 8.7).

Figure 8.7

Fenêtre de scénario de la scène principale.



Dans le calque actions, nous accédons au code suivant :

```
composantSousTitres.flvPlayback=ecranVideo;
composantSousTitres.source="gKaster/sousTitres.xml";
composantSousTitres.autoLayout=false;
composantSousTitres.captionTargetName="sousTitres_txt";
```

Dans le répertoire "gKaster" des exemples du livre, se trouve un document XML qui contient les sous-titres et repose sur la structure suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<tt xml:lang="fr" xmlns="http://www.w3.org/2006/04/ttaf1" xmlns:tts="http://www.w3.org
➤ /2006/04/ttaf1#styling">
  <head>
  </head>
  <body>
  <div xml:lang="fr">
    <p begin="00:00:00.00" dur="00:00:03.00">gKaster <span tts:fontFamily="Verdana"
      tts:fontSize="+12">C19</span></p>
    <p begin="00:00:09.00" dur="00:00:03.00">Le soleil se lève à l'horizon,</p>
    <p begin="00:00:12.45" dur="00:00:03.30">regarde, au bord du mont du village de
      paille.</p>
    <p begin="00:00:17.00" dur="00:00:02.30">Sur le fleuve fragile, dégoulinent,</p>
    <p begin="00:00:20.30" dur="00:00:02.30">marchent, les hommes en silence.</p>
    <p begin="00:00:25.00" dur="00:00:02.30">Sous le ciel gris du matin frais,</p>
    <p begin="00:00:28.30" dur="00:00:04.00">des pas lourds foulent le sol jonché
      des roses...</p>
    <p begin="00:00:36.00" dur="00:00:03.00">Des têtes s'y plient comme à la
      prière,</p>
    <p begin="00:00:40.00" dur="00:00:05.00">loin des terres arides s'est fait
      porter feu le poète.</p>
    <p begin="00:00:52.00" dur="00:00:05.00">Au travers de la clairière verdoyante,
      sous la rosée,</p>
    <p begin="00:01:00.30" dur="00:00:04.00">il a terminé ses années de douleur.</p>
    <p begin="00:01:06.00" dur="00:00:06.00">Mais quand le cercueil s'avance, noir,
      à travers la forêt verte du printemps</p>
    <p begin="00:01:13.00" dur="00:00:05.00">Le silence traverse la nature qui
      s'éveille.</p>
    <p begin="00:01:18.00" dur="00:00:04.00">Et là, le vent d'ouest s'arrête, en se
      demandant :</p>
    <p begin="00:01:22.30" dur="00:00:05.00">Qui foule ces roses d'un pas si
      lourd ?</p>
    <p begin="00:01:32.00" dur="00:00:06.00">Allez doucement, c'est peut-être une
      fleur qui vient de mourir.</p>
    <p begin="00:01:45.00" dur="00:00:06.00">Si j'étais Houragan, je
      l'accompagnerais jusqu'au bout du chemin.</p>
  </div>
  </body>
</tt>
```

Pour mettre en place un système de sous-titrage, il suffit de glisser-déposer une occurrence du composant FLVPlaybackCaptioning depuis la fenêtre des composants (Fenêtre > Composants) sur la même scène que celle où figure déjà le composant de lecture du flux vidéo FLVPlayback. Puis, il faut lui attribuer un nom d'occurrence. Dans notre exemple, nous le nommons composantSousTitres.

Pour lier l'objet `composantSousTitres` à la vidéo, dans le code, nous spécifions :

```
composantSousTitres.flvPlayback=ecranVideo;
```

Pour lui permettre d'identifier l'emplacement du fichier XML, nous ajoutons également :

```
composantSousTitres.source="gKaster/sousTitres.xml";
```

Nous avons la possibilité de magnétiser le champ de texte dynamique sur la vidéo (`true`) ou non (`false`). Pour que le champ de texte reste à sa position actuelle dans la scène, nous écrivons :

```
composantSousTitres.autoLayout=false;
```

Enfin, pour que le texte s'affiche dans un champ dynamique de notre propre création, nous associons, à une dernière commande, le nom d'occurrence du texte dynamique placé sur la scène :

```
composantSousTitres.captionTargetName="sousTitres_txt";
```

À défaut de spécifier un champ de texte dynamique pour accueillir les légendes, le composant les affichera directement sur la vidéo dans une zone rectangulaire de fond noir.

L'ensemble de ces instructions reprend les paramètres du composant `FLVPlaybackCaptioning` également disponibles et paramétrables depuis l'Inspecteur de composants.

Dans le fichier XML (de type texte TT – *Timed Text* – mais au format XML), nous pouvons lire une structure proche d'une page HTML. Ce document répond en fait à une organisation normalisée par le W3C (<http://www.w3.org/AudioVideo/TT/>). Tout en respectant la syntaxe propre à ce formatage, il nous suffit d'ajouter autant de ligne que de sous-titres doivent apparaître, et ensuite de renseigner chacune des propriétés.

Le fichier XML que nous utilisons repose sur le mécanisme suivant. Pour chaque nouveau sous-titre, le composant requiert une balise `<p></p>` :

```
<p begin="00:00:00.00" dur="00:00:03.00">gKaster
  <span tts:fontFamily="Verdana" tts:fontSize="+12">C19</span></p>
```

Dans cette balise, quatre attributs peuvent être renseignés : `begin`, `dur`, `end` et `style`. Dans notre exemple, les deux premiers seulement sont utilisés.

- L'attribut `begin` sert à définir le timecode à partir duquel le titre doit apparaître.
- L'attribut `dur` désigne la durée de ce sous-titre.
- L'attribut `end`, qui se définit de la même manière que les deux précédents, désigne le timecode de fin du sous-titre. Il n'a pas lieu d'être si l'on connaît déjà la durée.
- L'attribut `style`, optionnel, permet de gérer le formatage des textes à partir de styles HTML de base.

La gestion de l'affichage du sous-titre est précise, contrairement à la navigation utilisée avec la méthode `seek()`, car c'est le fichier XML qui détermine le moment où les textes doivent apparaître et non les images-clés de la vidéo.

Le dernier attribut, `style`, permet d'appliquer un style dont nous pouvons définir le formatage dans l'en-tête du document `<head></head>`, en amont. Pour ce faire, nous utilisons la syntaxe suivante :

```
<head>
  <styling>
    <style id="style1" tts:fontSize="20"/>
  </styling>
</head>
<body>
  <div xml:lang="en">
    <p begin="00:00:00.00" dur="00:00:03.00" style="style1">gKaster C19</p>
  </div>
</body>
```

Dans notre exemple, nous n'utiliserons pas les styles. En publiant le document Flash, la vidéo est lue et des légendes se succèdent les unes à la suite des autres, selon le timecode défini dans le fichier XML.

Le composant CaptionButton

Vous pouvez glisser-déposer le composant CaptionButton dans la même scène que le composant FLVPlaybackCaptioning. Il permet à l'utilisateur de désactiver et réactiver l'affichage des sous-titres. Ce composant, comme les boutons de contrôle de la vidéo, peut être personnalisé en double-cliquant dessus jusqu'à atteindre les objets graphiques qui le composent.

Étendre les formatages du document XML pour les sous-titres

Des options de formatage sont accessibles pour les données contenues dans le fichier XML des sous-titres. Ces formatages répondent à une norme précise et standardisée par le W3C. Vous trouverez le descriptif détaillé de ces options et les balises à employer à l'adresse suivante :

http://help.adobe.com/fr_FR/ActionScript/3.0/UsingComponentsAS3/WS5b3ccc516d4fbf351e63e3d118a9c65b32-7ee5.html.

Vidéo

À retenir

- Il est possible de déployer simplement un système de sous-titrage grâce à l'utilisation du composant FLVPlaybackCaptioning.
- La synchronisation avec le flux vidéo est précise car c'est le code qui gère l'affichage et non les images-clés de la vidéo.
- Des styles de formatage peuvent être appliqués aux textes des sous-titres, depuis Flash ou à partir de formatages définis dans le fichier XML.

Boucle vidéo

Par défaut, une vidéo gérée avec un composant FLVPlayback est lue de bout en bout sans boucler sur elle-même. Il peut être intéressant de permettre de relancer la vidéo pour créer un flux ininterrompu. Pour cela, nous utilisons une classe qui détecte le comportement de la vidéo. Cette classe se nomme videoEvent.

Dans cet exemple, nous utilisons une séquence vidéo associée à une barre de progression, ceci afin de nous permettre, lors de la publication, de tester l'effet de boucle plus rapidement, en déplaçant simplement la tête de lecture.

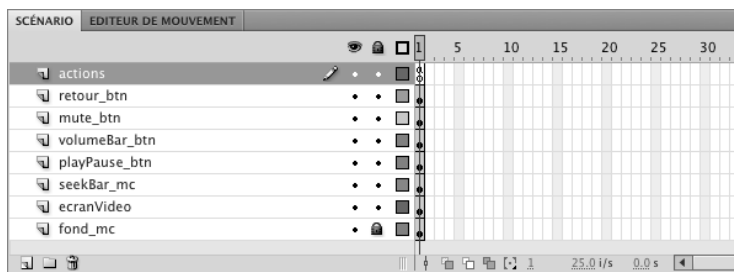


Exemples > ch8_videoInteractive_5.fla

Dans le document "ch8_videoInteractive_5.fla", la scène principale affiche un composant vidéo nommé `ecranVideo` et des occurrences de composants de contrôle réparties vers des calques distincts (voir Figure 8.8).

Figure 8.8

Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
import fl.video.VideoEvent;
//
ecranVideo.addEventListener(VideoEvent.COMPLETE,boucleVideo);
function boucleVideo(evt:VideoEvent) {
    écranVideo.seek(0);
    écranVideo.play();
}
```

D'abord, nous importons la classe `videoEvent` qui permet de détecter le comportement du flux vidéo. Puis, nous créons un écouteur que nous attachons au composant FLV-PlayBack placé sur la scène. Le gestionnaire d'événements fait référence à la classe importée et invoque la propriété `COMPLETE` afin de détecter le moment où la vidéo est terminée.

Lorsque la vidéo est achevée, la fonction `boucleVideo`, appelée par l'écouteur, exécute deux instructions. La première, `seek(0)`, replace la tête de lecture à l'image 0. La seconde, `play()`, indique de reprendre la lecture.

À retenir

- Il est possible de générer une boucle vidéo à l'aide de la classe `videoEvent`. Il faut pour cela remplacer la tête de lecture à l'image 0 et relancer la lecture de la vidéo.
- En détectant la fin de lecture de la vidéo, vous pouvez aussi associer d'autres actions, comme l'enchaînement avec d'autres flux vidéo ou l'affichage d'un autre contenu.

Enchaîner plusieurs vidéos à la suite

En détectant la fin de lecture d'un flux vidéo, vous êtes en mesure de placer d'autres instructions à la place du repositionnement de la tête de lecture comme nous l'avons fait ici. Par exemple, vous pouvez très bien invoquer un autre fichier vidéo en vue de créer un enchaînement de plusieurs vidéos.

Pour appeler une autre vidéo, utilisez l'instruction :

```
ecranVideo.source="cheminDeLaVideo.f4v" ;
```

Pensez éventuellement y adjoindre l'instruction `play()`, comme pour notre boucle, afin de garantir le redémarrage automatique de la lecture.

Synchroniser des actions avec les points de repère

Les points de repère (ou *CuePoints*) sont des marqueurs que l'on distribue tout au long de la vidéo. Ils permettent :

- D'y associer des actions de contrôle de lecture de la vidéo pour s'y référer comme dans un système de chapitrage (repères de navigation).
- D'y placer des événements programmés en ActionScript (repères d'événements), qui planifient des actions dans le temps.

Chacun des deux procédés requiert naturellement une vidéo, mais aussi la création des images-clés qui définissent le timecode sur lequel chaque repère doit être placé.

Il existe plusieurs techniques pour la création de ces repères : la première consiste à les intégrer physiquement dans l'encodage du signal vidéo, la deuxième à les stocker dans un fichier XML appelé ensuite avec ActionScript, la troisième consiste à les définir directement dans la fenêtre d'Actions du document par Flash et la quatrième engage le remplissage manuel de la propriété `cuepoints`, disponible depuis l'Inspecteur de composants.

- À l'intérieur du flux vidéo, les repères sont introduits depuis After Effects, Premiere Pro ou Adobe Media Encoder. Ils sont encapsulés dans le fichier une fois celui-ci rendu, si bien que toute modification du timecode, d'un seul de ces repères, induit de procéder à un nouveau rendu. Cette méthode présenterait peu d'intérêt si elle ne permettait pas un ciblage précis pour les repères de navigation. Dans ce contexte, en effet, à chaque fois que nous introduisons un repère de navigation, nous générons aussi une image-clé. Il en résulte que cette méthode est celle que nous préférons employer pour définir des repères de navigation, car ils permettent de travailler avec une grande précision. Pour les repères d'événements, en revanche, les autres techniques restent plus confortables.
- La création d'un fichier XML peut être intéressante pour les cas où un nombre élevé de repères d'événements est enregistré et où la valeur du timecode de chacun de ces repères doit être modifiée souvent. La gestion d'un fichier XML, dans ce cas précis, peut devenir intéressante car elle épargne d'avoir à republier le document Flash pour le mettre à jour. Nous n'abordons pas cette méthode dans notre ouvrage.

- La troisième option consiste à générer les points de repère d'événements directement dans la fenêtre d'actions du document Flash. Cette technique est très souple puisque, d'abord, le fait de dissocier les repères du flux vidéo va permettre de simplifier considérablement leur mise à jour. Ensuite, une ligne de code suffit par point de repère et, placés en tête de la fenêtre d'actions, toute mise à jour devient extrêmement simple à effectuer.
- La quatrième option, qui permet d'utiliser directement le composant, reprend le principe de la troisième, mais n'offre pas la souplesse de manipulation du code où nous pouvons plus facilement établir des relations entre le nom des repères que nous avons définis et les instructions à exécuter en regard des noms ajoutés. Nous n'abordons pas non plus cette dernière option dans notre ouvrage.

Dans cette section, nous décrivons la méthode de l'encodage dans le flux vidéo, pour l'utilisation des repères de navigation avec un ciblage précis. Plus loin, nous revenons sur un document Flash pour y introduire, en ActionScript, les repères d'événements dynamiques.

Repères de navigation

Dans cet exemple, nous allons placer des repères de navigation à l'intérieur de la vidéo demoreel de la société gKaster de manière à permettre une navigation précise. Cette vidéo est initialement encodée en F4V.

Les points de repère étant une fonctionnalité ajoutée par Adobe dans le format Flash vidéo, cette fonctionnalité n'est malheureusement utilisable que pour le format FLV. En plus de définir les points de repère, nous allons donc aussi modifier le format d'encodage de la vidéo dont nous disposons. Ce qui nous permet surtout d'aborder la manière d'encoder une vidéo pour l'ajout de repères de navigation. Naturellement, dans le cadre d'un projet réel de création de contenu, nous vous recommandons de repartir de la source vidéo non compressée pour obtenir un meilleur rendu.

Pour cet exemple, nous reprenons le dispositif de chapitrage utilisé préalablement avec la méthode `seek()`. Nous y remplaçons, après l'encodage, les références `seek()`, imprécises, par une action de détection de repères de navigation, plus précise, avec la méthode `seekToNavCuePoint()`.



Exemples > `ch8_videoInteractive_6 fla`

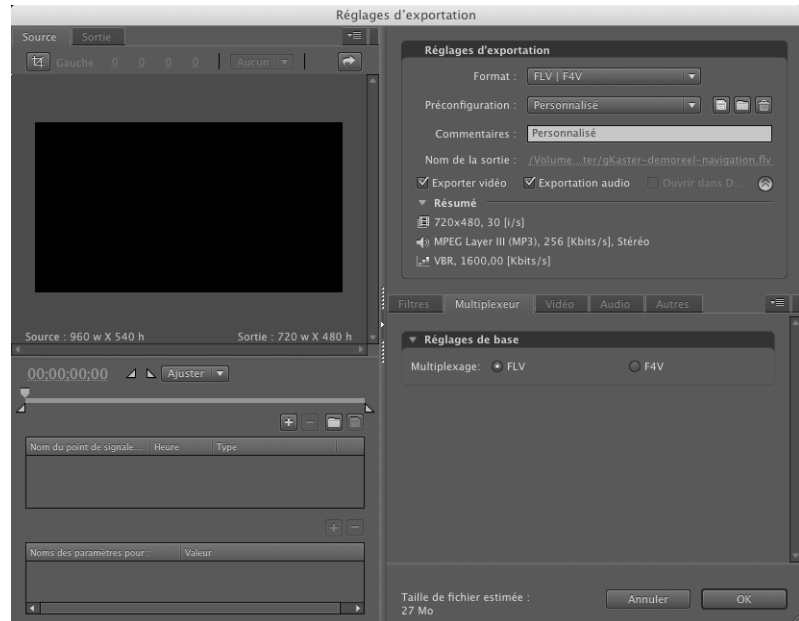
Encoder la vidéo

Revenons d'abord sur l'encodage de la vidéo :

1. Lancez l'application Adobe Media Encoder.
2. Ajoutez, à la liste de rendu, la vidéo intitulée `gKaster-demoreel.f4v`.
3. Puis, cliquez directement sur le lien jaune de la colonne **Prédéfinir**, ou bien sur le bouton **Réglages**, pour ouvrir la fenêtre de réglages (voir Figure 8.9).

Figure 8.9

Réglages
d'exportation.



4. Dans la partie de droite, sélectionnez l'onglet Multiplexeur puis activez l'option FLV.
5. Depuis l'onglet Vidéo, spécifiez la cadence de l'image. Dans notre exemple, nous maintenons une cadence d'origine à 30ips (voir Figure 8.10).

Figure 8.10

Choix de la
cadence avant
l'ajout de points
de repère.



Dans l'encodeur, notez qu'il est important de déterminer la cadence de la vidéo à partir du champ Image, de l'onglet vidéo, avant de créer les points de repère. Les points de repère sont ici calculés en fonction du nombre d'images par seconde préalablement défini. Si nous modifions la cadence de la vidéo après avoir généré les points de repère, comme nous configurons les points de repère à partir de données invariables, nous risquons de subir un décalage, voire, de ne pouvoir atteindre certaines séquences une fois le document Flash publié. Par exemple, nous spécifions une cadence initiale à 30ips, et nous ajoutons un repère à l'image 29 de la énième seconde. Si nous modifions, après avoir ajouté le repère, la cadence de la vidéo en la ramenant par exemple à 25ips, l'image 29 qui n'existe plus ne pourra être trouvée. Vérifiez ensuite que les dimensions de la vidéo correspondent à la surface disponible dans notre document (800 × 450 pixels).



Comment réactiver les champs de dimensionnement de l'encodeur ? L'encodeur étant un peu instable, vous pouvez avoir besoin de réactiver les options de réglage pour les éditer. Dans le volet de gauche de la fenêtre d'encodage, cliquez alors sur l'outil de recadrage, puis, dans l'onglet Sortie, sélectionnez l'option Modifier la taille de sortie. Puis, revenez dans l'onglet Source et désactivez l'outil de recadrage.

Ajustez enfin légèrement la compression de sorte que l'ajout d'images-clés ne réduise pas l'espace disponible pour coder les autres images, et donc, n'altère pas la qualité globale de la vidéo. Passez le débit minimum à 90.

Une fois l'échantillonnage calibré, nous pouvons ajouter les repères de navigation.

Ajout des repères de navigation

Pour placer des repères de navigation, nous devons revenir sur la partie gauche de la fenêtre d'encodage Adobe Media Encoder. Pour chaque repère, nous positionnons la tête de lecture à l'emplacement voulu. Puis, nous ajoutons un repère en cliquant, dans la partie inférieure, sur le bouton Plus.

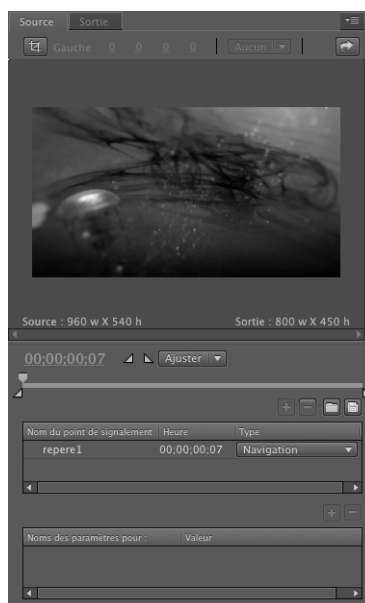
Placez la tête de lecture au timecode 00:00:00:07 qui correspond à la septième image. Pour contrôler la position de la tête de lecture avec précision, et glissez sur le timecode sans relâcher le pointeur. Attention un simple clic rend le champs éditable. Vous pouvez également y saisir manuellement une valeur, ou utiliser les flèches droite et gauche qui incrémentent ou décrémentent le time code de l'image. Si vous saisissez une valeur, pour confirmer son entrée, cliquez ensuite dans une zone neutre de la fenêtre. N'appuyez pas sur la touche Entrée qui ferme la fenêtre.

La vidéo étant initialement cadencée à 30 ips, le timecode s'arrête à l'image 30 et poursuit à la seconde suivante. La gestion du timecode ici n'est pas similaire à celle que nous avons contrôlée par ActionScript, qui est définie en secondes uniquement.

1. Dans la partie inférieure de la fenêtre, cliquez sur le bouton plus (+). Et, depuis le menu déroulant marqué Évènement, sélectionnez l'option Navigation.
2. Une entrée est enregistrée et ajoutée à la liste.
3. À gauche, cliquez sur le nom attribué par défaut, Point de signalement, pour le renommer. Saisissez par exemple : repere1 (voir Figure 8.11).

Figure 8.11

Ajout d'un repère de navigation.



Procédez de même pour l'ensemble des repères qui représentent le début de chaque séquence animée pour lesquelles nous disposons de vignettes dans le document Flash. Par exemple, placez la tête de lecture à l'image 00:00:13;13 pour ajouter une image-clé au début de la séquence qui correspond à la deuxième vignette dans Flash. Puis cliquez à nouveau sur Plus (+), pour définir un nouveau repère (voir Figure 8.12) et ainsi de suite.

Figure 8.12

Ajout d'un repère de navigation.



Les timecodes suivants correspondent à l'ensemble des séquences identifiées dans le Flash, par des vignettes. Une vidéo déjà codée avec les repères de navigation est disponible dans le dossier "gKaster" sous le nom "gKaster-demoreel-navigation.flv".

- 00;00;00;07 = repere1
- 00;00;13;14 = repere2
- 00;00;17;27 = repere3
- 00;00;41;14 = repere4
- 00;01;05;10 = repere5
- 00;01;29;26 = repere6
- 00;01;58;07 = repere7

Confirmez l'encodage en cliquant sur OK. Choisissez le nom de sortie "gkaster-demoreel-navigation.flv" et remplacez éventuellement le document déjà encodé qui porte le même nom. Puis lancez un rendu.

En plus des images-clés générées automatiquement toutes les deux secondes, chaque point de repère que nous avons spécifié en deviendra également une au moment de l'encodage. Nous pouvons maintenant réintégrer cette vidéo au document Flash et y saisir les instructions de navigation.

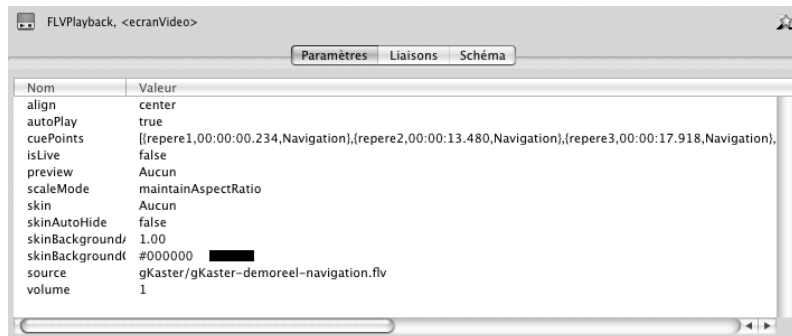
Détecter les points de repère avec ActionScript

Avant de programmer la détection des points de repère, nous devons mettre à jour le composant vidéo en y rattachant la vidéo exportée au format FLV. La vidéo étant déjà intégrée dans ce document, vous pouvez la remplacer en mettant simplement à jour la propriété source de l'Inspecteur de composants :

1. Revenez dans Flash et ouvrez le document "ch8_videoInteractive_6.fla".
2. Sélectionnez le composant vidéo.
3. Dans l'Inspecteur de composants (Fenêtre > Inspecteur de composants), modifiez la source en la cliquant deux fois. Puis, ciblez la vidéo que vous venez d'encoder et qui contient vos points de repère de navigation (voir Figure 8.13).

Figure 8.13

Liaison avec la vidéo.



Vous remarquez que l'Inspecteur de composants a détecté l'ensemble des repères que nous avons intégrés dans la vidéo, dans le paramètre `cuePoints`.

Revenez sur le calque actions. La fenêtre affiche le code suivant :

```
var repere:String="";

menu_mc.addEventListener(MouseEvent.CLICK,atteindreRepere);
function atteindreRepere(evt:MouseEvent) {
    if (evt.target.name=="lien1_btn") {
        repere="repere1";
    }
    if (evt.target.name=="lien2_btn") {
        repere="repere2";
    }
    if (evt.target.name=="lien3_btn") {
        repere="repere3";
    }
    if (evt.target.name=="lien4_btn") {
        repere="repere4";
    }
    if (evt.target.name=="lien5_btn") {
        repere="repere5";
    }
    if (evt.target.name=="lien6_btn") {
        repere="repere6";
    }
    if (evt.target.name=="lien7_btn") {
        repere="repere7";
    }
    trace(repere)
    ecranVideo.seekToNavCuePoint(repere);
}
```

De la même manière que nous avons distribué la méthode `seek()` dans les chapitres précédents, nous utilisons ici la méthode `seekToNavCuePoint()`. Nous spécifions, en paramètre, une valeur "chaîne de caractères" qui correspond au nom du point de repère de navigation ajouté dans l'encodeur, et ce, pour chaque lien.

Publiez le document en faisant **Cmd+Entrée** (Mac) ou **Ctrl+Entrée** (Windows). La vidéo joue instantanément. En cliquant sur chacune des vignettes, le ciblage atteint précisément le timecode spécifié dans le flux vidéo.

Repères d'événements

Les repères d'événements ont pour objectif de planifier une action à l'intérieur du document Flash, à mesure de la progression de la vidéo.

Comme pour les repères de navigation, nous devons au préalable définir les repères d'événements avant d'ajouter les actions. Pour créer les repères, nous pouvons les placer dans le flux vidéo, comme vu pour la navigation ou bien les coder dynamiquement en **ActionScript**.

L'avantage à le coder repose sur l'idée suivante : un repère d'événements, par rapport au repère de navigation, est indépendant du flux vidéo. Si un repère de navigation doit être dans la vidéo, parce qu'il cible une image-clé précise de la vidéo, le repère d'événements, lui, vise une action, par nature indépendante de la vidéo. Les repères d'événements n'ont donc aucune raison d'être encodés dans le flux vidéo. En somme, les repères d'événements agissent un peu comme des drapeaux (*flags* ou *labels*), des déclencheurs lus par la tête de lecture en même temps que l'image à laquelle ils sont attachés, et ce, quelle que soit la teneur du flux vidéo. Il devient donc plus judicieux de les traiter en ActionScript, et non en dur dans la vidéo. Ainsi, nous pouvons utiliser une vidéo au format F4V, de meilleure qualité, et non se limiter uniquement au format FLV, qui est le seul à gérer les points de repère.

Dans cette section, nous allons voir comment associer à un composant qui cible une vidéo enregistrée en F4V, des repères codés en ActionScript. Dans notre exemple, les actions que nous associons aux repères permettent de repositionner un MovieClip blanc sous la vignette qui correspond à la séquence en cours de visualisation. À mesure que la vidéo progresse, la vignette est repositionnée, à l'aide d'une interpolation de type TweenMax, sur la vignette suivante, et ce, jusqu'à la dernière.



Exemples > ch8_videoInteractive_7.fla

Le document "ch8_videoInteractive_7.fla" contient un composant qui appelle la vidéo "gKaster-demoreel.f4v". Sous le composant, figure un menu composé de MovieClip possédant tous un nom d'occurrence. L'un d'entre eux, situé en arrière-plan, signale la progression de la lecture. Il est placé sous la première image qui matérialise la première séquence du flux vidéo (voir Figure 8.14 et 8.15).

Figure 8.14

Aperçu du document.

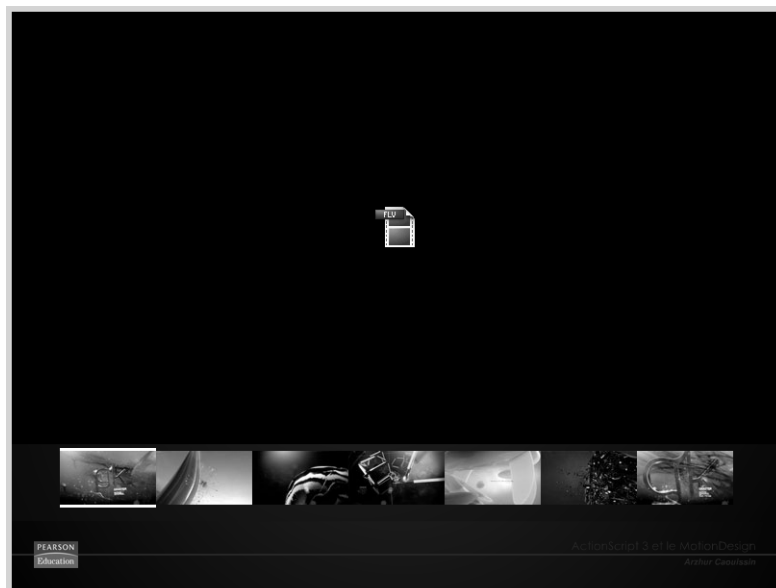
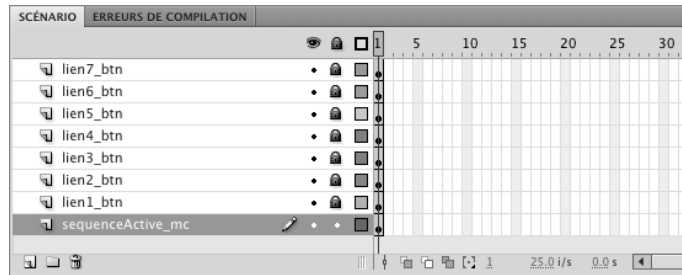


Figure 8.15

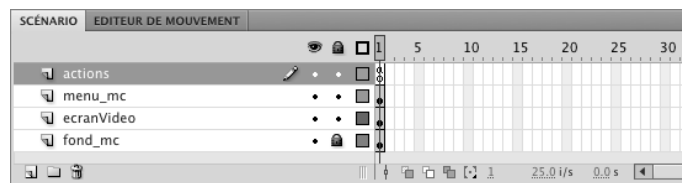
Fenêtre de scénario du symbole menu.



Dans le scénario, au-dessus du calque fond_mc, nous retrouvons le composant et le menu (voir Figure 8.16).

Figure 8.16

Fenêtre de scénario de la scène principale.



Dans la fenêtre Actions, apparaît le code suivant :

```
// importation des classes
import fl.video.MetadataEvent;
import gs.*;
import gs.easing.*;
import gs.events.*;

// création des points de repère
ecranVideo.addASCuePoint(0,"repere1");
ecranVideo.addASCuePoint(13.14,"repere2");
ecranVideo.addASCuePoint(17.27,"repere3");
ecranVideo.addASCuePoint(41.14,"repere4");
ecranVideo.addASCuePoint(65.10,"repere5");
ecranVideo.addASCuePoint(89.26,"repere6");
ecranVideo.addASCuePoint(118.07,"repere7");

// traitement des actions
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,ecouteRepere);
function ecouteRepere(evt:MetadataEvent) {
    if (evt.info.name=="repere1") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-300, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere2") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-200, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere3") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-100, ease:Strong.easeInOut});
    }
}
```

```

    }
    if (evt.info.name=="repere4") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:0, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere5") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:100, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere6") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:200, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere7") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:300, ease:Strong.easeInOut});
    }
    trace(ecranVideo.playheadTime)
}

```

En premier, nous importons les classes nécessaires. La classe `MetadataEvent` permet de gérer les métadonnées liées à un contenu vidéo. Les classes `Greensock gs` permettent de créer les animations `TweenMax`, comme vu au Chapitre 2 :

```

// importation des classes
import fl.video.MetadataEvent;
import gs.*;
import gs.easing.*;
import gs.events.*;

```

À la suite, nous définissons les points de repère, un à un. Il y en a sept. Chacun d'entre eux est structuré ainsi :

```
ecranVideo.addASCuePoint(0,"repere1");
```

Nous ciblons d'abord le composant auquel ces repères doivent être associés avec le nom du composant vidéo (`ecranVideo`). Puis, nous ajoutons un repère ActionScript avec la méthode `addASCuePoint`, méthode à l'intérieur de laquelle deux paramètres sont attendus dont le timecode (en secondes) et son nom (une chaîne de caractères). Décliné pour chaque vignette, nous obtenons :

```

// création des points de repère
ecranVideo.addASCuePoint(0,"repere1");
ecranVideo.addASCuePoint(13.14,"repere2");
ecranVideo.addASCuePoint(17.27,"repere3");
ecranVideo.addASCuePoint(41.14,"repere4");
ecranVideo.addASCuePoint(65.10,"repere5");
ecranVideo.addASCuePoint(89.26,"repere6");
ecranVideo.addASCuePoint(118.07,"repere7");

```

Une fois les repères ajoutés, il reste à y associer des actions. Nous plaçons pour ce faire un écouteur sur le composant `ecranVideo`, composant sur lequel nous venons d'attacher les repères en ActionScript. Nous faisons directement référence aux points de repère véhiculés par cette classe, avec la propriété `CUE_POINT` :

```

// traitement des actions
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,ecouteRepere);
function ecouteRepere(evt:MetadataEvent) {
    // actions sur chaque point de repère
}

```

À l'intérieur du bloc d'instruction de la fonction, nous plaçons les actions à exécuter à chaque fois que l'écouteur détecte un point de repère, quel qu'en soit le nom, quelle qu'en soit

la nature (événement ou navigation). Pour distinguer une action d'une autre, nous utilisons une structure conditionnelle qui permet de spécifier l'action à conduire en fonction du nom de chaque point de repère. Ce qui donne :

```
// traitement des actions
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,ecouteRepere);
function ecouteRepere(evt:MetadataEvent) {
    if (evt.info.name=="repere1") {
        TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-300, ease:Strong.easeInOut});
    }
}
```

Si, le nom de l'objet (`evt.info.name`), identifié par l'écouteur, correspond à une certaine valeur, alors, dans cet exemple, une interpolation de type `TweenMax` est exécutée et déplace le MovieClip blanc à une abscisse déterminée, qui correspond à la position courante de la vignette. Nous déclinons ce principe pour chacun des points de repère.

À la fin de la fonction, une instruction permet de vérifier la position du timecode au moment où l'action est exécutée, avec la méthode `playheadTime`, que nous avons déjà abordée pour gérer l'accélération et le rembobinage en début de chapitre :

```
trace(ecranVideo.playheadTime)
```



Nous avons utilisé dans cette section une instruction pour générer dynamiquement des points de repère et permettre l'utilisation du format F4V. Mais si la vidéo était au format FLV et contenait ses propres repères, fussent de navigation, la fonction que nous avons développée fonctionnerait également. Les instructions liées à la création des points de repère en ActionScript seraient simplement inutiles. La fonction, en effet, que les repères soient inclus dans les métadonnées de la vidéo ou créés dynamiquement en ActionScript, agit de la même manière.

À retenir

- Il est possible de créer un système de navigation précis en utilisant des points de repère de navigation encodés dans une vidéo FLV.
- Pour naviguer dans des points de repère encodés dans les métadonnées de la vidéo, nous utilisons la méthode `seekToNavCuePoint()` ;.
- Les points de repère encodés dans un flux F4V ne sont pas pris en compte.
- Il est possible de gérer des événements ActionScript sur une vidéo F4V à partir de points de repère ajoutés dynamiquement avec ActionScript.
- Il est possible de générer des points de repère depuis l'Inspecteur de composants, dans l'onglet paramètres, avec la propriété `cuepoints`, de la même manière que nous le ferions dans Adobe Media Encoder.

Lire une vidéo en arrière

Lire une vidéo Flash en arrière avec fluidité peut être intéressant dans le cadre d'une présentation interactive graphiquement élaborée. Par exemple, un fabricant de voitures, un

bagagiste de luxe, un joaillier, peuvent vouloir valoriser leurs créations dans une mise en scène fine réalisable uniquement en vidéo (lumières, détails, animation 3D avec radiosit  et v locit , etc.). En permettant alors   l'utilisateur de prendre la main sur le flux vid o pour le laisser se d placer librement autour de l'objet film  ou mod lis , celui peut contr ler le sens de d filement de la t te de lecture dans la vid o, comme s'il naviguait dans un espace r el.

Tant que l'utilisateur d place la t te de lecture vers l'avant, la vid o reste fluide. Mais, lire un flux vid o en arri re aboutit g n ralement   une lecture saccad e de l'image, quelle que soit le type de programmation utilis , ceci m me avec des fichiers encod s avec 100 % d'images-cl s, m me en jouant avec deux fichiers dont un projet    l'endroit et l'autre   l'envers. Le repositionnement al atoire et de vitesse variable du pointeur, contr l  par l'utilisateur, implique une autre solution.

La solution la plus fluide et la plus simple   d ployer pour g rer une lecture vid o avant et arri re, consiste   int grer le flux vid o physiquement dans le document Flash et   en contr ler la lecture   l'aide d'un gestionnaire de type `ENTER_FRAME`. Pour rendre la lecture plus fluide, nous mettons  galement   jour l'affichage   chaque it ration.

L'int gration d'une vid o   l'int rieur d'un document Flash appelle cependant quelques recommandations :

- La vid o doit de pr f rence  tre encod e en FLV et avec un nombre d'images-cl s int gral (1 image-cl  par image). Le FLV semble en effet plus permissif   la lecture arri re sur des images-cl s pleines que ne l'est un F4V.
- L'int gration d'une vid o dans Flash est techniquement limit e   16 000 images, mais elle doit proposer un temps de chargement honorable. Donc, pensez   placer, en amorce du document, une jauge de chargement qui avertisse l'utilisateur sur sa progression (voir Chapitre 5).
- Au-del  de 2 Mo de poids pour votre document Flash, vid o comprise, r duisez les dimensions de la vid o. Diminuez sa dur e. Compressez davantage son d bit.
- Nous ajoutons la m thode `updateAfterEvent()` pour fluidifier la lecture de la sc ne.

Dans cette section, nous utilisons une vid o FLV qui repr sente une galaxie en mouvement. La s quence est courte et dure 179 images (soit 7,16 secondes pour un flux   25ips). Une fois compress e avec une image-cl  par image, le fichier ne p se que 1,4 Mo. Dans cet exemple, un curseur est positionn  sur l' cran pour que l'utilisateur prenne la main sur le contenu et naviguer comme bon lui semble   l'int rieur du flux vid o.



Exemples > `ch8_videoInteractive_8 fla`

Dans le document "`ch8_videoInteractive_8 fla`", un `MovieClip` intitul  `vid o_mc` contient une s quence vid o import e. La s quence vid o est  tendue int gralement dans le sc nario de ce `MovieClip`. Sur la sc ne principale, le symbole `console_mc` contient un autre `MovieClip` nomm  `curseur_mc` (voir Figure 8.17).

Figure 8.17

Aperçu du document.



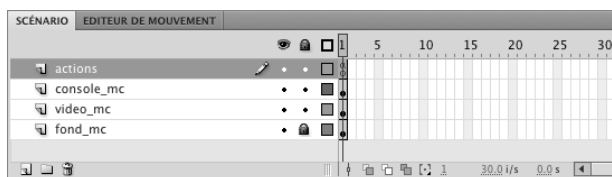
Info

Pour intégrer physiquement une vidéo dans le scénario, faites Fichier > Importer > Importer de la vidéo. Puis, cliquez sur Parcourir pour sélectionner le fichier vidéo déjà compressé au format FLV. Sélectionnez l'option Incorporer le fichier FLV dans le SWF et le diffuser dans le scénario. Puis cliquez sur Continuer. Choisissez alors de l'inclure de préférence dans un clip. Puis, confirmez jusqu'à la fermeture de la boîte de dialogue.

Dans le scénario, au-dessus du calque `fond_mc`, nous pouvons identifier les objets clairement répartis sur des calques distincts (voir Figure 8.18).

Figure 8.18

Fenêtre de scénario.



Le calque actions affiche le code suivant :

```
//----- initialisation
video_mc.stop();
var pourcentage:Number;
var positionTeteDeLecture:Number=0;
var initMouseX:Number=console_mc.curseur_mc.x;

//----- scroll
var zoneDeDeplacement:Rectangle = new Rectangle(10,0,780,0);

console_mc.curseur_mc.addEventListener(MouseEvent.MOUSE_DOWN, activerDeplacement);
```

```

function activerDeplacement(evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME, jouerVideo);
    console_mc.cursueur_mc.startDrag(false, zoneDeDeplacement);
    evt.updateAfterEvent();
}

addEventListener(MouseEvent.MOUSE_UP, stopperDeplacement);
function stopperDeplacement(evt:MouseEvent) {
    evt.currentTarget.stopDrag();
    removeEventListener(Event.ENTER_FRAME, jouerVideo);
}

//----- jouer la vidéo
function jouerVideo (evt:Event) {
    pourcentage=Math.ceil((console_mc.cursueur_mc.x-initMouseX)/(stage.stageWidth/100));
    positionTeteDeLecture=Math.ceil(pourcentage*(video_mc.totalFrames/100))-10;
    video_mc.gotoAndStop(positionTeteDeLecture);
}

```

Pour bien comprendre la mécanique du curseur, nous devons d'abord considérer que le clip qui accueille la vidéo comporte un certain nombre d'images (179). De même, le curseur qui se déplace dans la console est mobile sur une certaine largeur (de 10 à 780 pixels). Nous devons donc, comme nous l'avons fait pour l'ascenseur au Chapitre 2, créer une équation qui convertit la position du curseur en numéro d'image à atteindre.

Le programme démarre avec, dans les premières lignes du code, une instruction qui interrompt la lecture du clip vidéo, avec un stop. Nous initialisons ensuite deux variables nombre que nous utilisons plus loin pour le calcul du rapport entre la position du curseur et celle de la tête de lecture à l'intérieur du clip vidéo. La troisième variable, nommée `initMouseX` enregistre la position de départ du curseur, en X :

```

//----- initialisation
video_mc.stop();
var pourcentage:Number;
var positionTeteDeLecture:Number=0;
var initMouseX:Number=console_mc.cursueur_mc.x;

```

Plus loin, nous définissons les actions qui permettent de gérer le défilement du curseur le long de l'axe représenté, dans notre document, par un filet blanc :

```

//----- scroll
var zoneDeDeplacement:Rectangle = new Rectangle(10,0,780,0);

console_mc.cursueur_mc.addEventListener(MouseEvent.MOUSE_DOWN, activerDeplacement);
function activerDeplacement(evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME, jouerVideo);
    console_mc.cursueur_mc.startDrag(false, zoneDeDeplacement);
    evt.updateAfterEvent();
}
addEventListener(MouseEvent.MOUSE_UP, stopperDeplacement);
function stopperDeplacement(evt:MouseEvent) {
    evt.currentTarget.stopDrag();
    removeEventListener(Event.ENTER_FRAME, jouerVideo);
}

```

La dernière commande de la fonction `activerDeplacement` utilise la méthode `updateAfterEvent()`. Cette méthode rafraîchit la scène immédiatement après que les autres

instructions ont été exécutées, à savoir, dès que le déplacement du curseur a eu lieu. Cela rend le déplacement de l'objet plus fluide que si nous limitons le rafraîchissement de l'affichage à un taux basé uniquement sur la cadence de la scène (avec `ENTER_FRAME`). De même, grâce à cette méthode, l'effet de déplacement dans la vidéo est un peu plus fluide.

Toujours dans la fonction `activerDeplacement`, une instruction appelle la fonction `jouerVideo`, détaillée plus bas, qui contrôle la position de la tête de lecture dans le clip vidéo.

Dans la fonction `stopperDeplacement`, nous arrêtons le déplacement du curseur (`stopDrag`) ainsi que le déplacement de la tête de lecture dans le clip vidéo (`removeEventListener`).

Plus bas, la fonction appelée au déplacement du curseur (`jouerVideo`) calcule la position de la tête de lecture, dans le clip `video_mc` :

```
//----- jouer la vidéo
function jouerVideo (evt:Event) {
    pourcentage=Math.ceil((console_mc.curseur_mc.x-initMouseX)/(stage.stageWidth/100));
    positionTeteDeLecture=Math.ceil(pourcentage*(video_mc.totalFrames/100))-10;
    video_mc.gotoAndStop(positionTeteDeLecture);
}
```

Dans la fonction `jouerVideo`, la valeur `pourcentage` ramène d'abord sous la forme d'un pourcentage, la position courante du curseur. Mais, au préalable, nous prenons soin de retrancher la valeur véhiculée par la variable `initMouseX`, avant la division. Cela nous permet d'initialiser le pourcentage à la valeur zéro, et non à la valeur exacte de positionnement du curseur, qui n'est pas placé à zéro au démarrage de l'application, mais à 10 pixels dans notre exemple.

La deuxième ligne du bloc d'instruction reprend cette valeur et la multiplie par un coefficient. Ce coefficient transpose le pourcentage, obtenu au-dessus, à l'échelle de la durée de la vidéo. Pour définir ce coefficient, nous prenons le nombre d'images contenues dans le clip vidéo (`totalFrames`) et le divisons par 100. Le numéro d'une image de scénario étant un chiffre entier, nous utilisons la classe `Math.ceil()` pour arrondir ce chiffre. L'image zéro n'existe pas dans un scénario. Cette méthode permet donc aussi d'obtenir une valeur supérieure à zéro et d'atteindre une image qui existe, en toute circonstance.

Une fois la valeur calculée, elle est transmise en paramètre de la méthode `gotoAndStop()` qui contrôle la position de la tête de lecture dans le clip vidéo.

À retenir

- Pour réaliser une interface où l'utilisateur prend la main sur le flux vidéo, sans sautillerment ni rupture de flux, nous importons la vidéo dans le Flash en respectant les contraintes de poids qui permettent de préserver l'accessibilité du contenu.
- Pour que le défilement soit fluide, nous contrôlons le rafraîchissement de la scène avec la méthode `updateAfterEvent()`.
- Pour que le flux d'image ne saute pas, le fichier vidéo doit être encodé en FLV avec 100 % d'images-clés.

Arrêter une vidéo dans un SWF imbriqué

Un site bien conçu conduit généralement à isoler les rubriques dans des SWF séparés et à ne les importer que s'ils ont été invoqués par l'utilisateur. Dans ce contexte, si vous débutez en ActionScript 3, vous avez peut-être éprouvé des difficultés à vider un chargeur de son contenu, d'une part, mais surtout à interrompre l'audio d'un fichier vidéo ou d'une musique, que la rubrique chargée pouvait contenir. Lorsque vous supprimez de la liste d'affichage (`removeChild`) un chargeur, vous ne supprimez pas son contenu comme nous le faisons en AS2 – avec `unloadMovie()` –, vous le "désaffichez". Pour contourner cette difficulté, quatre solutions sont désormais disponibles :

- Vous pouvez interrompre le contenu d'un chargeur depuis le SWF qui a appelé le contenu avec une série d'instructions compatibles Flash 9.
- Vous pouvez procéder à la même manipulation avec une simple et unique instruction compatible Flash 10.
- Vous pouvez décharger et interrompre le SWF appelé depuis le SWF lui-même, avec des instructions compatibles Flash 9.
- Vous pouvez décharger et interrompre le SWF appelé depuis le SWF lui-même, avec quelques instructions compatibles Flash 10.

Dans cette section, nous abordons les quatre solutions pour interrompre les contenus. Mais nous détaillons également quelques notions chères aux anciens codeurs AS2, le ciblage des contenus imbriqués et l'accès aux contenus de la scène parente ou racine.

Dans cet exemple, nous disposons d'un premier SWF qui en appelle un autre, lequel joue une vidéo. Nous avons placé, à l'intérieur de chaque document, un bouton Fermer qui permet, chacun dans son contexte respectif et individuellement, de supprimer le contenu chargé, et cela avec les deux syntaxes Flash 9 et Flash 10.



Exemples > `ch8_videoInteractive_9 fla` et Exemples > `ch8_videoInteractive_9b fla`

Dans le document "`ch8_videoInteractive_9 fla`", sur la scène, un `MovieClip` nommé `cible_mc` sert de conteneur pour le SWF chargé. Au-dessus, un contour blanc représente l'espace qu'occupe le contenu de ce SWF une fois celui-ci importé (voir Figure 8.19).

Dans la fenêtre de scénario, les symboles `cible_mc` et le bouton Fermer sont répartis distinctement vers les calques (voir Figure 8.20).

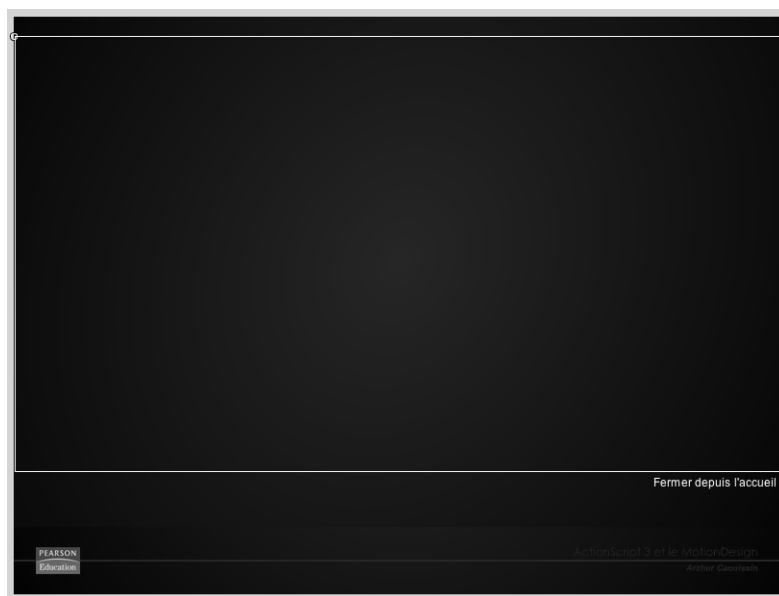
Dans le second document, nommé "`ch8_videoInteractive_9b fla`", la scène affiche seulement un composant et un bouton de fermeture, localisé (voir Figure 8.21).

Dans la fenêtre de scénario du document chargé, le composant et le bouton fermer sont également répartis vers les calques (voir Figure 8.22).

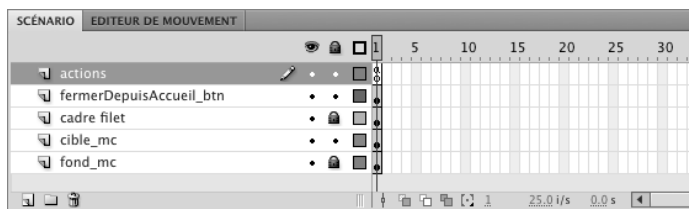
La structure des deux documents permet de décharger et stopper les contenus aussi bien depuis le document racine que depuis le document appelé.

Figure 8.19

Aperçu du SWF de départ.

**Figure 8.20**

Fenêtre de scénario.

**Figure 8.21**

Aperçu du document SWF chargé.

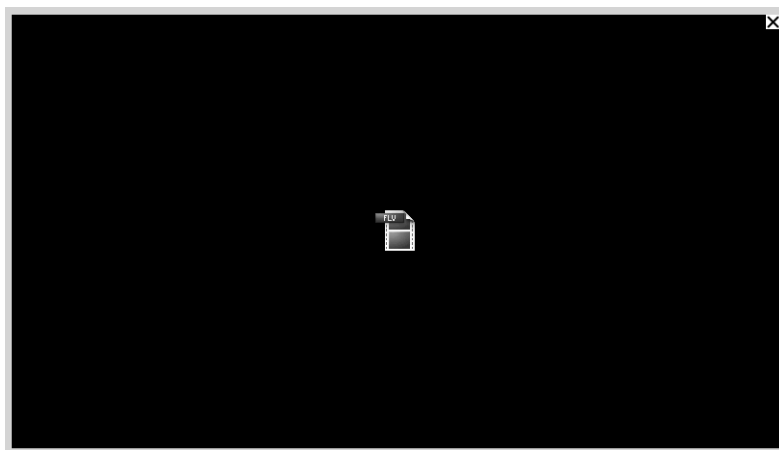
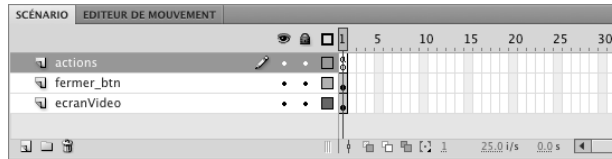


Figure 8.22

Fenêtre de scénario du document chargé.



Arrêt et fermeture depuis le document racine

Dans le calque actions du document racine ("ch8_videoInteractive_9 fla"), celui qui appelle l'autre, nous lisons le code suivant :

```
//----- chargement

var chemin:URLRequest = new URLRequest("ch8_VideoInteractive_9b.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);

chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);

function afficher(Evt:Event){
    cible_mc.addChild(chargeur);
}

//----- fermer depuis l'accueil

fermerDepuisAccueil_btn.addEventListener(MouseEvent.CLICK, fermerSWF);
function fermerSWF (evt:MouseEvent) {
    /*
    //Méthode avant Flash 10
    MovieClip(chargeur.content).ecranVideo.stop();
    chargeur.unload();
    */

    chargeur.unloadAndStop();
}
```

Dans ce code, la première étape consiste à charger le second document SWF qui exécute la vidéo. Pour cela, nous définissons un nouveau chargeur, lequel exécute l'affichage du contenu une fois celui-ci chargé (voir Chapitre 5 pour le détail de ces instructions) :

```
//----- chargement

var chemin:URLRequest = new URLRequest("ch8_VideoInteractive_9b.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);

chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);

function afficher(Evt:Event){
    cible_mc.addChild(chargeur);
}
```

Plus loin, nous ajoutons un écouteur sur le bouton nommé `fermer_btn`. Dans la fonction appelée par l'écouteur, nous distinguons deux types d'instructions. La première,

laissée en commentaire pour ne pas doubler celle qui est restée active, est compatible Flash 9 :

```
MovieClip(chargeur.content).ecranVideo.stop();  
chargeur.unload();
```

Si nous nous contentions de purger le chargeur, le contenu serait bien ôté de l'interface, mais le flux ne serait pas stoppé, un peu comme si le robinet continuait à déverser de l'eau alors que l'on ait retiré le sceau. Le son et l'image continuent de se lire. Cette commande consiste donc, dans un premier temps, à interrompre tout ce qui est exécuté. En l'occurrence, la vidéo. Puis, sur la deuxième ligne, à purger le chargeur.

Pour cibler plus spécifiquement le contenu chargé dans un SWF, nous devons d'abord faire référence au chargeur que nous avons utilisé pour importer le contenu. Plus précisément, nous devons faire référence à ce qu'il contient, et non à l'enveloppe qu'il représente. Nous spécifions donc, en plus de reprendre son nom (chargeur), que nous ciblons son contenu (chargeur.content).

Ensuite, la référence au contenu du chargeur est passée en paramètre d'une méthode de transtypage en MovieClip. Et la deuxième ligne, une fois la vidéo arrêtée, purge le chargeur de son contenu.

Pourquoi transtyper un chargeur en MovieClip ?

Il faut comprendre que chaque document AS3 possède désormais une structure où la scène principale n'est plus un simple MovieClip (comme ce fut le cas en AS1 ou en AS2), mais bien un objet de type Sprite, qui contient la scène principale MovieClip. Une couche intermédiaire a donc été ajoutée à la structure d'un document Flash. Si cette modification structurelle a permis d'apporter une plus grande stabilité et de déployer de nombreuses fonctionnalités dans Flash, elle ne permet plus de cibler directement un contenu chargé dans un SWF imbriqué. Chaque document est réellement indépendant. Le Sprite ne possède pas les mêmes propriétés de ciblage que MovieClip. Pour transgresser cette contrainte, il faut donc transtyper le Sprite qui contient la scène MovieClip du document importé, en MovieClip. Ainsi, toute la chaîne d'imbrication préserve une structure intégralement composée de MovieClip imbriqués, et autorise de nouveau le ciblage. Pour en savoir plus sur la structure native d'un document Flash en AS3 et sur le principe de la liste d'affichage, consultez la documentation Adobe à la page suivante : http://livedocs.adobe.com/flash/9.0_fr/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts_bak&file=00000142.html.

Cette première méthode, compatible Flash 9, peut vite devenir astreignante si les contenus chargés deviennent multiples. C'est la raison pour laquelle, à la demande des utilisateurs, une nouvelle instruction est apparue avec Flash 10 : `unloadAndStop()`.

```
chargeur.unloadAndStop();
```

Concernant le ciblage du contenu chargé, le principe reste donc le même. Mais une seule instruction suffit désormais pour interrompre tous les contenus et purger le SWF. Voici la liste des actions désormais associées à cette nouvelle méthode :

- Les flux audio et vidéo sont arrêtés.

- Les écouteurs d'événements sont supprimés de la scène.
- Les écouteurs d'événements `enterFrame`, `frameConstructed`, `exitFrame`, `activate` et `deactivate` sont supprimés.
- Les timers (horloges) sont arrêtées.
- Les occurrences `Camera` et `Microphone` sont retirées.
- Les clips sont stoppés.

Arrêt et fermeture depuis le contenu chargé

Une autre technique, pour fermer un contenu, consiste à placer les actions directement dans le SWF chargé.

Dans le calque actions du document appelé ("ch8_videoInteractive_9b fla"), contenant la vidéo, nous lisons ce code-ci :

```
//----- fermer depuis le SWF chargée

fermer_btn.addEventListener(MouseEvent.CLICK, fermerSoiMeme);
function fermerSoiMeme (evt:MouseEvent) {
    /*
    //Méthode avant Flash 10
    ecranVideo.stop();
    MovieClip(root.parent.root).chargeur.unload();
    */

    MovieClip(root.parent.root).chargeur.unloadAndStop() ;

    //MovieClip(this.stage.getChildAt(0)).chargeur.unloadAndStop();
}
```

Dans ce code, un écouteur est attaché à un bouton situé sur la scène principale, localement. Cet écouteur appelle une fonction qui propose deux types d'instructions : celles compatibles Flash 9 et Flash 10.

Les deux premières instructions sont en commentaire, pour de nouveau éviter de doubler l'instruction active :

```
ecranVideo.stop();
MovieClip(root.parent.root).chargeur.unload();
```

La première ligne interrompt d'abord le flux vidéo contenu dans la scène courante, comme nous le ferions simplement à travers une action classique.

La deuxième reprend le principe de ciblage du contenu dans le chargeur, évoqué précédemment. Mais ce chargeur ayant été créé dans la scène du document parent, nous le ciblons en y faisant référence par `transtypage`.

En paramètre de la méthode `MovieClip()`, nous indiquons simplement le chemin qui nous permet d'accéder au contenu ciblé, avec `root.parent.root`.

Ciblage dans un document Flash AS3

Avec la structure Sprite d'un document AS3, `root`, qui autrefois désignait la racine globale de l'ensemble des documents Flash imbriqués (d'équivalent `_root` en AS1 et en AS2), ne cible désormais plus que la scène courante.

Il est en revanche toujours possible de remonter d'un niveau en utilisant la propriété `parent`, de deux niveaux en utilisant `parent.parent`, et ainsi de suite. Mais pour ce faire, vous devez préciser de quel objet vous partez, en ajoutant à la position de départ et celle d'arrivée un `root`. Vous obtenez alors `root.parent.parent.root`, pour cibler la scène courante d'un conteneur de conteneur `parent`.

En ActionScript 3, la racine globale, qui correspondait avant à `_root`, est désormais désignée par la propriété `stage` de l'objet conteneur que constitue le document racine. Pour l'invoquer, nous écrivons à présent `this.stage.getChildAt(0)`.

Voir aussi la note suivante pour un exemple de ciblage avec `this.stage.getChildAt(0)`.

Exemples de ciblage de contenus dans des SWF imbriqués (AS2 et AS3)

En ActionScript 2, pour cibler un contenu dans un autre SWF, nous procédions ainsi :

- Cibler un symbole intitulé `clip_mc` dans un SWF chargé, contenu dans `cible_mc` :
`cible_mc.clip_mc.rotation=90;`
- Cibler un symbole intitulé `clip_mc` dans un SWF parent, contenu dans `cible_mc` :
`_parent.cible_mc.clip_mc.rotation=90;`
- Ou bien, si le SWF parent est le conteneur principal : `_root.cible_mc.clip_mc.rotation=90;`
- Pour cibler un contenu intitulé `clip_mc` et placé deux conteneurs en amont, nous inscrivons :
`_parent._parent.cible_mc.clip_mc.rotation=90;`

En ActionScript 3, pour cibler un contenu dans un autre SWF, nous procédons désormais ainsi :

- Cibler un symbole intitulé `clip_mc` dans un SWF chargé, contenu dans `cible_mc`. Le nom du conteneur n'apparaît plus. Nous invoquons directement le chargeur :
`MovieClip(nomDuChargeur.content).clip_mc.rotation=90;`
- Cibler un symbole intitulé `clip_mc` dans un SWF parent, contenu dans `cible_mc` :
`MovieClip(root.parent.root).cible_mc.clip_mc.rotation=90;`
- Ou bien, si le SWF parent est le conteneur principal (équivalent en AS3 de `_root`) :
`MovieClip(this.stage.getChildAt(0)).cible_mc.clip_mc.rotation=90;`
- Pour cibler un contenu intitulé `clip_mc` et placé deux conteneurs en amont, nous inscrivons :
`MovieClip(root.parent.parent.root).cible_mc.clip_mc.rotation=90;`

À défaut d'utiliser la méthode `getChildAt()`, vous pouvez également cibler un objet en utilisant `getChildByName()`. Mais vous devez alors connaître le nom de l'objet pour le passer en paramètre de cette méthode. Ce qui donne, par exemple :

```
getChildByName("clip_mc");
```

Dans notre exemple, la fonction propose aussi une instruction compatible Flash 10 :

```
MovieClip(root.parent.root).chargeur.unloadAndStop();  
//MovieClip(this.stage.getChildAt(0)).chargeur.unloadAndStop();
```

Comme énoncé plus haut, nous ciblons ici le chargeur situé dans le SWF parent. Puis, nous reprenons, comme pour le lien placé dans le SWF parent, la méthode `unloadAndStop()`.

Une variante de la première ligne est ajoutée en commentaire. Elle propose une alternative à la syntaxe `root.parent.root`, pour cibler directement la scène du conteneur principale (avec `this.stage.getChildAt(0)`), que nous aurions autrefois appelée avec `_root`.

En publiant les deux documents, le premier charge le second et exécute directement la vidéo. Lorsque vous cliquez sur le lien situé en haut et à droite de la vidéo, comme sur le lien situé sous la vidéo à droite, celle-ci est d'abord arrêtée, puis disparaît (voir Figure 8.23).

À retenir

- Il est possible de cibler des objets dans des documents imbriqués et, inversement, de se référer à des objets placés dans un document racine, en ActionScript 3. Pour cela, nous transtypons les chargeurs avec la méthode `MovieClip()` et ciblons leur contenu avec la propriété `content`.
- Pour supprimer un document imbriqué qui joue une vidéo, il faut d'abord interrompre le flux, puis purger le chargeur qui a appelé ce contenu.
- Une nouvelle méthode, en ActionScript 3, permet d'interrompre les contenus chargés et de purger le chargeur, simultanément : `unloadAndStop()`.

Figure 8.23

Aperçu du document.



Synthèse

Dans ce chapitre, vous avez appris à contrôler l'interactivité sur la vidéo en créant vous-même vos propres commandes pour, entre autres, lire, stopper, accélérer et rembobiner la vidéo et modifier le volume sonore avec progression. Vous avez appris à réaliser un système de sous-titrage en utilisant un affichage de texte personnalisé, à créer différents dispositifs de chapitrage. Vous avez appris à réaliser des présentations vidéo fluides y compris quand elles sont lues en arrière et à vitesse variable. Vous avez appris à gérer l'intégration de contenus vidéos dans des documents imbriqués et à contourner la complexité de l'Action-Script 3 pour arrêter les contenus audio et vidéo chargés. D'une manière plus générale, vous avez aussi appris à maîtriser le ciblage de contenus dans une structure de site complexe. Vous êtes à présent en mesure de réaliser des sites richmédia impliquant l'organisation de contenus vidéos et une couche importante d'interactivité.

9

La 3D native

Introduction

La gestion de la 3D est possible nativement dans Flash depuis la version CS4. Mais elle reste toutefois limitée à l'animation d'objets 2D dans un espace 3D. Dès lors que l'utilisateur dispose d'un lecteur Flash récent (Flash 10), il peut l'exécuter. Les propriétés 3D des objets du scénario, couplées à des interpolations programmées en ActionScript (avec la classe `TweenMax`), ouvrent de nouvelles "perspectives" dans la conception de sites Internet.

Dans ce chapitre, nous allons voir comment programmer des animations en 3D, dans Flash, de manière simple et accessible. Pour cela, nous utilisons en premier exemple un livre avec des pages qui tournent – un flipbook –, puis, nous analyserons des systèmes de présentation de contenus avec différents types de galeries 3D, impliquant entre autres des images et de la vidéo.

À l'issue de ce chapitre, vous saurez créer des interfaces 3D conviviales, en toute simplicité.

Les propriétés 3D disponibles nativement pour l'animation dans Flash sont les suivantes :

- `x` positionne l'objet sur l'axe des abscisses.
- `y` positionne l'objet sur l'axe des ordonnées.
- `z` positionne l'objet sur l'axe Z.
- `rotationX` effectue une rotation de l'objet sur l'axe des abscisses.
- `rotationY` effectue une rotation de l'objet sur l'axe des ordonnées.
- `rotationZ` effectue une rotation de l'objet sur l'axe Z.

Ces propriétés permettent de déplacer et pivoter les objets dans l'espace de la même manière que nous le faisons avec d'autres propriétés, y compris à travers des animations programmées ou non.

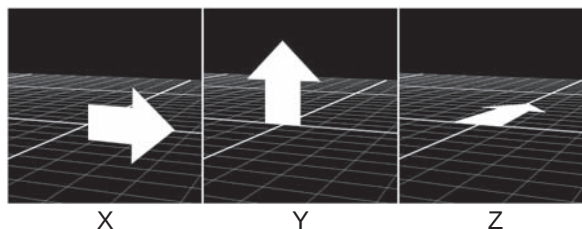


Dans Flash, l'axe X représente la ligne horizontale qui part de la gauche vers la droite. L'axe Y, la ligne qui descend verticalement. L'axe Z, enfin, désigne la ligne frontale qui part du premier plan (l'écran) et se dirige vers l'infini en arrière-plan (voir Figure 9.1).

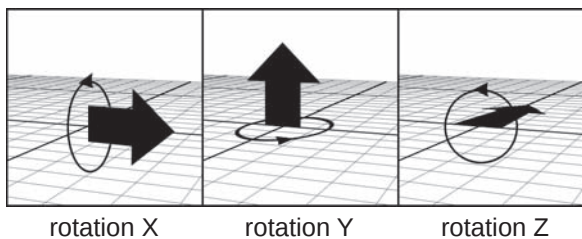
La rotation dans l'espace 3D se détermine par rapport aux axes X, Y et Z. Une rotation en X fait pivoter un objet sur l'axe X tout en le maintenant dans sa direction à la manière du balai d'une moissonneuse (voir Figure 9.2). Une rotation en Y fait pivoter un objet autour de l'axe des ordonnées (axe vertical) à la manière des lames d'un mixeur. Une rotation en Z applique un mouvement circulaire sur l'axe frontal des profondeurs, à la manière des ailes d'un moulin à vent que nous observerions de face.

Figure 9.1

Définition des axes X, Y et Z d'un espace 3D.

**Figure 9.2**

Définition des rotations X, Y et Z dans un espace 3D.

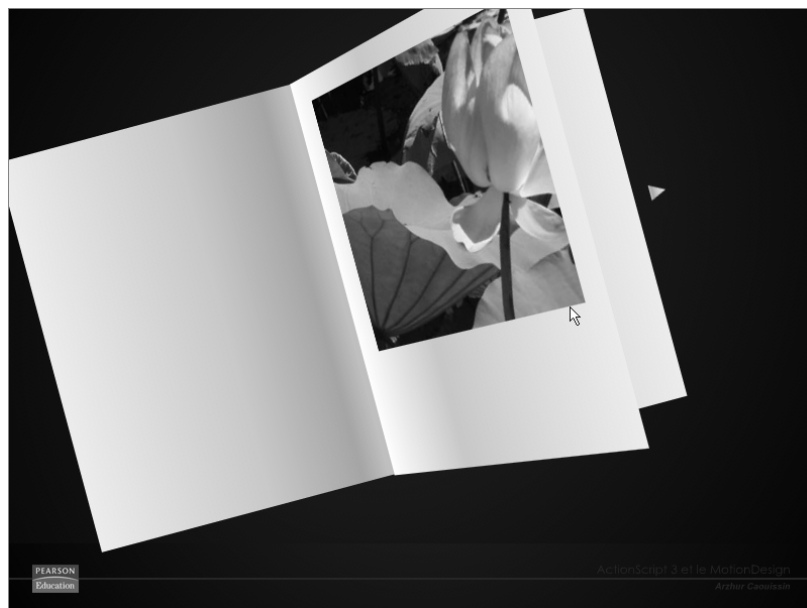


Animer un livre 3D en ActionScript

Dans cette section, nous allons aborder la conception d'un livre avec des pages qui tournent. Pour cela, nous utilisons la classe TweenMax, compatible avec les propriétés 3D de Flash ainsi que des propriétés de rotation dans l'espace (voir Figure 9.3).

Figure 9.3

Aperçu après publication.



En publiant le document, une animation fait entrer le livre au centre de l'écran. En cliquant sur la flèche située à droite du livre ou sur le livre lui-même, la page tourne sur un axe 3D et laisse apparaître la page suivante, ceci pour toutes les pages. Une fois le livre terminé, une animation 3D le retire de l'écran en le faisant pivoter dans l'espace 3D, sur lui-même.



Exemples > ch9_3DNative_1.fla

Dans le document "ch9-3DNative-1.fla", sur la scène principale, au-dessus du calque fond_mc, apparaît un symbole de type MovieClip nommé livre_mc (voir Figures 9.4 et 9.5). Le livre est placé hors champ.

Figure 9.4

Aperçu de la scène principale.

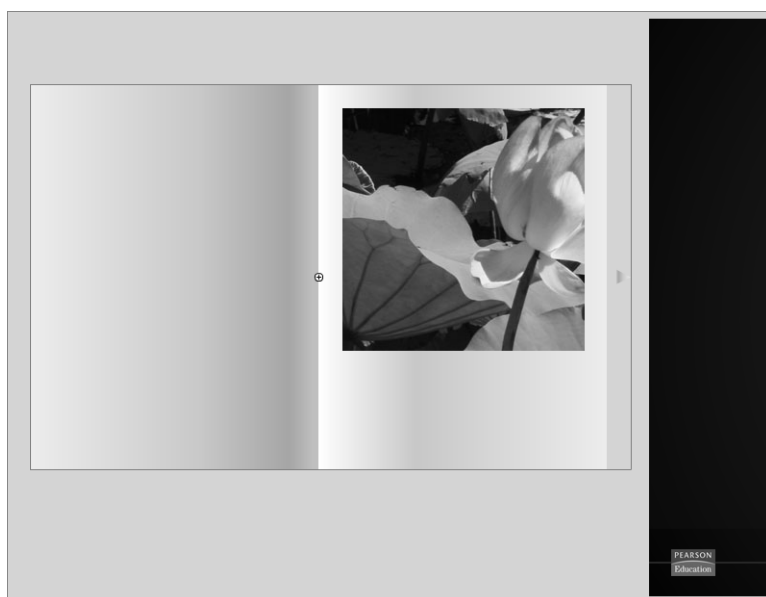


Figure 9.5

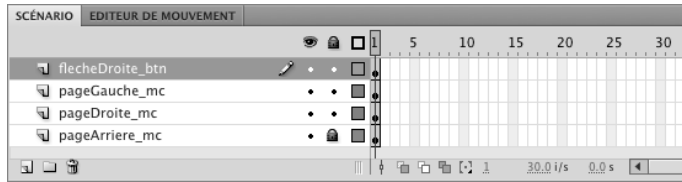
Aperçu du scénario de la scène principale.



Le MovieClip livre_mc contient quatre symboles. Le premier représente la page de gauche (livre ouvert), c'est-à-dire, la page de couverture. Un autre clip représente la page de droite et comporte l'ensemble des écrans de toutes les pages du livre. En effet, une seule page est animée et elle laisse donc derrière elle un vide à chaque action. Une page Arrière sert de fond pendant les transitions. Enfin, un bouton en forme de flèche active le mécanisme des pages qui tournent (voir Figure 9.6).

Figure 9.6

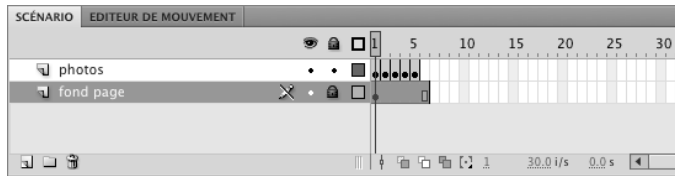
Aperçu du scénario du symbole pageDroite_mc.



Le symbole pageDroite, ayant pour nom d'occurrence pageDroite_mc, contient une série d'écrans qui constituent les différentes pages du livre. Chaque écran est distribué dans le scénario sous la forme d'images-clés, placées les unes à la suite des autres. La dernière image est vide. Elle représente le verso de la page tournée (voir Figure 9.7).

Figure 9.7

Aperçu du scénario du symbole pageDroite_mc.



Sur la scène principale, au sommet de tous les calques, figure le calque Actions (voir Figure 9.5). Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var tweenPage:TweenMax;
var nombreDePages:Number=livre_mc.pageDroite_mc.totalFrames-1;
var i:Number=1;

livre_mc.pageDroite_mc.stop();

//----- animation d'introduction
var tween3D:TweenMax=TweenMax.to(livre_mc,4,{x:330, rotationZ:-15, delay:0,
  ➤ ease:Strong.easeInOut});
TweenMax.from(livre_mc.pageGauche_mc,2,{rotationY:-180, delay:2,
  ➤ ease:Strong.easeInOut});
TweenMax.from(livre_mc.flecheDroite_btn,2,{alpha:0,delay:4,
  ➤ ease:Strong.easeInOut});

//----- cliquer pour tourner les pages
livre_mc.addEventListener(MouseEvent.CLICK,pageSuivante);

function pageSuivante (evt:MouseEvent) {
    i++;
    // page de garde
    if (i<=1) {
        livre_mc.swapChildren(livre_mc.pageGauche_mc,livre_mc.pageDroite_mc);
    }
    // page de droite
    if (i<=nombreDePages) {

        tween3D=TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
```

```

tween3D.addEventListener(TweenEvent.COMPLETE, tween3DFini);
} else {
    //animation de fermeture
    livre_mc.pageArriere_mc.visible=false;

    TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
    TweenMax.to(livre_mc,4,{rotationZ:360, x:1000, z:-500, alpha:0,
    ➡ rotationY:360, delay:1, ease:Strong.easeInOut});
    livre_mc.flecheDroite_btn.visible=false;
    tweenPage.addEventListener(TweenEvent.COMPLETE, tweenPageFini);
}
}

function tween3DFini (evt:TweenEvent) {
    livre_mc.pageDroite_mc.gotoAndStop(i);
    TweenMax.from(livre_mc.pageDroite_mc,1,{alpha:0,ease:Strong.easeOut});
    livre_mc.pageDroite_mc.rotationY=0;
}

function tweenPageFini (evt:TweenEvent) {
    removeEventListener(Event.ENTER_FRAME,masquerVersoPage);
}

//----- masquer le verso des pages qui tournent
addEventListener(Event.ENTER_FRAME,masquerVersoPage);
function masquerVersoPage (evt:Event) {
    if (livre_mc.pageDroite_mc.rotationY>90) {
        livre_mc.pageDroite_mc.gotoAndStop(6);
    }
}

```

Ce code est structuré en quatre parties : l'initialisation, l'animation d'introduction, les actions des pages qui tournent et la gestion du verso des pages tournées.

La première partie du code importe les classes utilisées pour les transitions TweenMax :

```

import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

```

À la suite, nous définissons quelques variables. La première identifie une interpolation TweenMax en vue de lui faire succéder d'autres actions :

```

var tweenPage:TweenMax;

```

La deuxième compte le nombre de pages du livre :

```

var nombreDePages:Number=livre_mc.pageDroite_mc.totalFrames-1;

```

La méthode totalFrames, abordée au Chapitre 8, permet ici de compter le nombre total d'images dans le scénario du symbole pageDroite_mc. Nous lui retranchons cependant 1 afin de ne pas comptabiliser la page vide utilisée pour le verso.

Nous définissons ensuite une variable i, utilisée pour l'incréméntation des pages, à chaque fois qu'une page est tournée. La valeur démarre à 1, car la première image du scénario de pageDroite_mc démarre aussi à 1 et, bien que masquée, est déjà active.

```

var i:Number=1;

```

Enfin, nous stoppons le défilement automatique de la tête de lecture dans le symbole `pageDroite_mc`. À défaut, nous les verrions toutes défiler jusqu'à la première interaction. Ceci équivaut à placer un `stop` directement en tête de la première image de ce même clip :

```
livre_mc.pageDroite_mc.stop();
```

Les premières instructions consistent à réaliser une animation d'introduction pour placer le livre en position de lecture, prêt à être ouvert. Nous déployons pour cela, dans cette deuxième partie, quelques transitions de type `TweenMax` :

```
//----- animation d'introduction
var tween3D:TweenMax=TweenMax.to(livre_mc,4,{x:330, rotationZ:-15, delay:0,
➤ ease:Strong.easeInOut});
TweenMax.from(livre_mc.pageGauche_mc,2,{rotationY:-180, delay:2,
➤ ease:Strong.easeInOut});
TweenMax.from(livre_mc.flecheDroite_btn,2,{alpha:0,delay:4,
➤ ease:Strong.easeInOut});
```

Dans ces interpolations, nous plaçons, en paramètre, des propriétés 3D (`rotationZ`, `rotationY`). Nous utilisons les propriétés 3D de la même manière que les autres propriétés (`x`, `alpha`, `scaleX`, etc.).

La première occurrence d'animation est typée. C'est pour nous permettre de disposer d'un nom d'objet, par la suite, sur lequel nous pourrions placer un écouteur et enchaîner avec d'autres actions (voir Chapitre 2).

Dans notre animation, le livre vient se positionner dans le champ en s'inclinant légèrement à -15° (`rotationZ:15`). Cette transition dure quatre secondes et démarre à l'instant 0.

Puis, dès la deuxième seconde, dans la deuxième interpolation, la page de gauche du livre pivote sur son axe Y (`rotationY:-180`) et révèle la page de droite placée à l'arrière-plan. Cette transition dure deux secondes et se termine en même temps que la première. Les effets des deux animations se superposent dans le temps. Enfin, une transition fait apparaître la flèche droite et signale l'interactivité. Les enchaînements de ces transitions ont été gérés avec la propriété `delay`, sans recours aux écouteurs.



L'accumulation de filtres sur des objets animés en 3D, avec des textures dégradées et des animations en alpha, sollicite beaucoup les ressources graphiques. Si des sautilllements apparaissent, pensez à réduire ces effets.

Dans la troisième partie, un écouteur est attaché au livre. Il appelle la fonction `pageSuivante` :

```
//----- cliquer pour tourner les pages
livre_mc.addEventListener(MouseEvent.CLICK,pageSuivante);

function pageSuivante (evt:MouseEvent) {
// instructions des pages qui tournent.
}
```

Nous ciblons directement le livre, et non pas seulement la flèche contenue dans le livre, afin de permettre à l'utilisateur d'activer le changement de page même en cliquant sur le livre. Puisque la flèche est contenue dans le symbole du livre, le script fonctionnera également en

cliquant sur la flèche. En proposant une action plus étendue, nous renforçons l'ergonomie du projet.

À l'intérieur du bloc d'instruction de la fonction, nous commençons par incrémenter *i* :

```
i++;
```

Puis, nous vérifions sa valeur. Si cette valeur est inférieure au nombre de pages à visiter, alors, nous intervertissons, dans la liste d'affichage, les index de position des symboles `pageGauche` et `pageDroite` :

```
// page de garde
if (i<=1) {
    livre_mc.swapChildren(livre_mc.pageGauche_mc,livre_mc.pageDroite_mc);
}
```

La méthode `swapChildren()` appelle en paramètres les deux objets, séparés par une virgule, à substituer dans la liste d'affichage (`livre_mc.pageGauche_mc`, `livre_mc.pageDroite_mc`). Le premier objet prend ainsi l'index du second.

Nous procédons à l'interversion des objets car l'animation 3D se matérialise sur le niveau de l'objet uniquement, et non sur l'ensemble de la scène. La scène n'est pas réellement projetée en 3D. Seuls les objets le sont individuellement. Des propriétés générales existent pour la scène, mais les transformations des objets se font localement, à chaque niveau. Si bien que lorsque nous faisons pivoter un objet dans l'espace, si celui-ci est placé sous un symbole, l'objet restera toujours en arrière-plan. Pour contourner ce problème, nous replaçons au premier plan (`pageGauche`), celui placé à l'arrière-plan (`pageDroite`).

Une fois la valeur de *i* vérifiée et l'index de la page de droite placé au premier plan, nous déployons le mécanisme de rotation :

```
// page de droite
if (i<=nombreDePages) {

    tween3D=TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
    tween3D.addEventListener(TweenEvent.COMPLETE, tween3DFini);
}
```

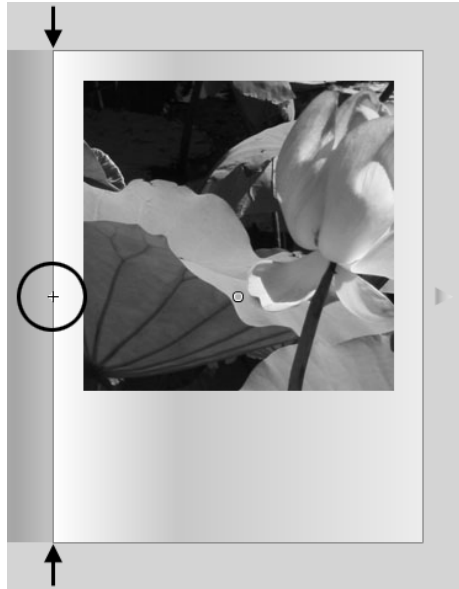
Nous programmons la rotation de la page à l'aide de la propriété `rotationY` et d'une interpolation de type `TweenMax`. La page tourne donc bien sur sa verticale. Enfin, pour nous assurer que l'objet pivote sur sa tranche et non en son centre, nous avons placé le centre du symbole à gauche, dès sa création (voir Figure 9.8).

Un écouteur est attaché au nom d'objet de l'interpolation `TweenMax` pour nous permettre de lancer une autre action, une fois l'animation terminée. Cette fonction, qui est exécutée quand la page est tournée, déplace la tête de lecture dans le symbole `pageDroite` et replace la page tournée à sa position initiale :

```
function tween3DFini (evt:TweenEvent) {
    livre_mc.pageDroite_mc.gotoAndStop(i);
    TweenMax.from(livre_mc.pageDroite_mc,1,{alpha:0,ease:Strong.easeOut});
    livre_mc.pageDroite_mc.rotationY=0;
}
```


Figure 9.8

La position du centre détermine l'axe de rotation.



Nous avons également ajouté une interpolation qui affiche la page avec un fondu de type alpha, allant de 0 à 1 (avec from) :

```
TweenMax.from(livre_mc.pageDroite_mc,1,{alpha:0,ease:Strong.easeOut});
```

Nous devons comprendre que le fait de tourner la page va nécessairement produire un vide à l'emplacement où cette page figurait avant de pivoter. Pour combler ce trou, nous plaçons, à l'arrière-plan, le symbole pageArriere_mc. Afin que la nouvelle page contenant l'image n'apparaisse pas brutalement, nous utilisons un fondu en alpha.

Il est possible d'afficher plus tôt le contenu des pages en multipliant les symboles page-Droite dans le livre, d'une part, et en multipliant d'autant les instructions gotoAndStop dans le programme.

Si la condition pour réaliser la rotation de la page n'est pas vérifiée (if (i<=nombreDePages)), ou si le nombre de pages visitées a atteint la limite autorisée, alors else, une autre condition, est exécutée :

```
else {
    //animation de fermeture
    livre_mc.pageArriere_mc.visible=false;

    TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
    TweenMax.to(livre_mc,4,{rotationZ:360, x:1000, z:-500, alpha:0,
    ➡ rotationY:360, delay:1, ease:Strong.easeInOut});
    livre_mc.flecheDroite_btn.visible=false;
    tweenPage.addEventListener(TweenEvent.COMPLETE, tweenPageFini);
}
```

Lorsque les images de la page de droite sont toutes visitées, la page arrière est d'abord masquée (`visible=false`) car il ne reste plus de page à tourner. La scène devient visible à l'arrière-plan du livre.

Une fois le livre refermé, deux transitions sont exécutées. La première fait pivoter la dernière page. La seconde fait s'envoler le livre, mais avec un délai d'une seconde. Ceci permet au livre de se refermer avant de le voir s'envoler.

Les mouvements du livre sont composés d'une rotation sur les axes Z (`rotationZ:360`) et Y (`rotationY:360`), puis d'un déplacement en X et Z. La valeur utilisée pour le déplacement du livre, en Z, est négative (`z:-500`). Cela signifie que l'objet quitte la scène tout en se rapprochant de l'écran.

Puis, nous masquons le bouton flèche pour éviter que l'utilisateur ne relance l'action. Nous signalons aussi la fin du programme.

Enfin, pour d'optimiser les ressources de l'utilisateur, nous invoquons une fonction qui interrompt, à la fin de la transition, le gestionnaire `ENTER_FRAME`, désormais inutile (`tweenPageFini`).

Dans les quatrième et dernière parties, pour terminer, nous avons ajouté une fonction qui masque le verso des pages tournées :

```
//----- masquer le verso des pages qui tournent
addEventListener(Event.ENTER_FRAME,masquerVersoPage);
function masquerVersoPage (evt:Event) {
    if (livre_mc.pageDroite_mc.rotationY>90) {
        livre_mc.pageDroite_mc.gotoAndStop(6);
    }
}
```

Nous spécifions que, dès que la page tournée atteint une rotation Y de 90°, c'est-à-dire dès qu'elle apparaît sur la tranche et que l'on ne perçoit plus son contenu, au recto ni au verso, nous déplaçons, avec un gestionnaire de type `ENTER_FRAME`, la tête de lecture de l'objet page à l'image 6, qui représente une page vide. Sans cette fonction, nous pourrions voir des deux côtés du symbole la même image, mais inversée. En affichant ponctuellement, juste le temps de la rotation, l'image 6, nous donnons l'illusion de pages indépendantes les unes des autres.

Pour compléter, l'action appelée par le gestionnaire `ENTER_FRAME` fait référence à une nouvelle image du symbole `pageDroite` afin de compenser le fait que le moteur 3D affiche les contenus sur les deux faces simultanément. Lorsque la page bascule à 90°, il nous suffit donc de modifier l'image affichée dans ce clip pour donner l'illusion que le verso se distingue du recto. Ainsi, pour créer un livre recto-verso, vous pouvez ajouter dans le symbole `pageDroite_mc` autant de pages verso que voulues, puis les appeler depuis le gestionnaire `ENTER_FRAME` en fonction de `i`, à travers la même méthode `gotoAndStop()`. Vous obtenez :

```
livre_mc.pageDroite_mc.gotoAndStop(i+nombreDePages);
```

Pensez dans ce cas à adapter aussi le calcul de la variable `nombreDePages`, en fonction du nombre de pages verso. Par exemple :

```
Var nombreDePages:int = (livre_mc.pageDroite_mc.totalFrames-1) / 2
```



Réaliser un livre interactif avec InDesign. Des systèmes très réalistes de livres interactifs préprogrammés (composants pageFlip) existent sur le Web et sont facilement accessibles. Dans la suite Adobe, par exemple, InDesign CS4 intègre cette solution. Vous pouvez exporter tout type de mise en pages en livre Flash interactif au format SWF. Il ne reste plus, ensuite, qu'à lier le document à votre site avec un chargeur programmé en ActionScript, comme nous l'avons vu précédemment dans le chapitre sur les galeries d'images. Pour exporter une mise en pages sous la forme d'un livre réaliste au format SWF, depuis InDesign, faites Fichier > Exporter. Dans les options d'exportation, choisissez SWF. Pour aller plus loin dans la gestion de documents mis en pages, reportez-vous au Chapitre 15, section "Gérer le PDF".

Lisser les images pour la 3D. L'affichage 3D peut générer un crénelage sur les images lorsqu'elles sont projetées dans un espace 3D ou simplement inclinées. Pour lisser les images, depuis la fenêtre de Bibliothèque, sur chaque image, faites clic-droit > Propriétés. Puis, dans la fenêtre de dialogue, activez l'option Autoriser le lissage. Vous pouvez procéder à la même action en programmation. Reportez-vous à la section "Lissage des images bitmap", au Chapitre 11, pour connaître les solutions de lissage en programmation.

3D avec la classe Tween

Les propriétés 3D natives de Flash peuvent être gérées avec la classe Tween. Pour un objet clip_mc animé, par exemple, avec les propriétés z et rotationX, vous obtenez ceci :

```
import fl.transitions.Tween;
import fl.transitions.easing.*;
var monTween:Tween;
monTween = new Tween(clip_mc, "z", Elastic.easeInOut, 0, 500, 3, true);
var monTweenRx:Tween;
monTweenRx = new Tween(clip_mc, "rotationX", Elastic.easeInOut, 0, 500, 3, true);
```

3D avec la classe Caurina

Les propriétés 3D natives de Flash peuvent être gérées avec la classe Caurina (disponible sur <http://code.google.com/p/tweener/downloads/list>). Pour un objet clip_mc animé avec, par exemple, les propriétés z et rotationX, vous obtenez ceci :

```
import caurina.transitions.Tweener;
Tweener.addTween(clip_mc, {x:300, y:300, z:330, rotationX:90, rotationY:0,
➤ rotationZ:0, time:5, transition:"easeinoutexpo"});
```

3D avec la classe TweenMax

Les propriétés 3D natives de Flash peuvent être gérées avec la classe TweenMax. Pour un objet clip_mc animé avec, par exemple, les propriétés z et rotationX, vous obtenez ceci :

```
import gs.TweenMax;
import gs.easing.*;
TweenMax.to(clip_mc, 2, {rotationX:90, rotationY:90, z:500, delay:1,
➤ ease:Bounce.easeOut});
```

À retenir

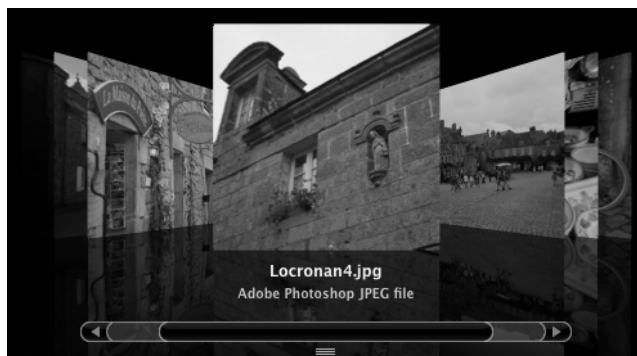
- Flash autorise la gestion de la 3D y compris à travers les interpolations des classes TweenMax, Caurina et Tween. Il suffit d'y introduire les propriétés 3D x, y, z, rotationX, rotationY et rotationZ.
- Pour animer un objet en 3D, celui-ci doit être un MovieClip.
- La gestion de la 3D dans Flash est localisée sur chaque objet. Il faut donc modifier l'ordre d'apparition des objets dans la liste d'affichage lorsque leurs mouvements se croisent.
- La position du centre de chaque symbole animé en 3D est déterminant pour la définition des mouvements dans l'espace.

Carrousel d'images 3D

Un carrousel d'images 3D est une suite d'images que l'on peut faire défiler dans un espace 3D. Les images qui passent au centre sont frontales. Celles qui restent en retrait de la zone centrale se repositionnent dans l'espace 3D par une transition animée. Les carrousels d'images peuvent être linéaires, circulaires ou épouser d'autres formes. L'exemple le plus illustre de ce type de présentation est le système de navigation intitulé *CoverFlow* et développé par Apple. On le retrouve principalement sur le système Mac OS X, dans l'iPhone, iTunes et de nombreux services associés à la marque Apple (voir Figure 9.9).

Figure 9.9

Apple CoverFlow.



Dans cette section, nous allons voir comment faire défiler des images sur une bande horizontale. Mais nous présentons surtout comment définir un repositionnement 3D automatique, en fonction de la position de chaque image sur l'axe des abscisses X.

Pour cet exemple, nous proposons deux documents codés différemment, selon que vous préférez un codage simple à partir d'objets placés manuellement dans le scénario ou un codage plus adapté à un interfaçage dynamique (pour lier la présentation à un fichier XML, par exemple).

Version simplifiée

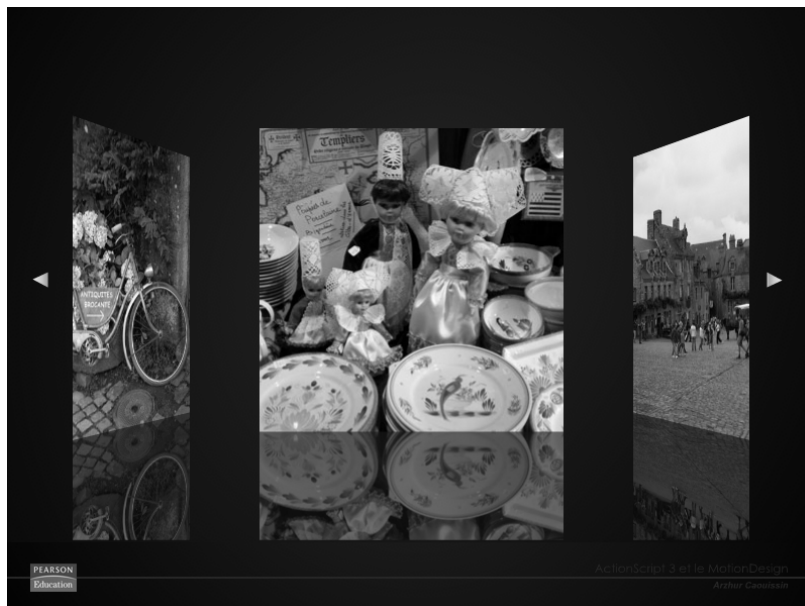


Exemples > ch9_3DNative_2.fla

En publiant le premier document, nommé "ch9_3DNative_2.fla", deux flèches donnent accès au défilement d'une suite d'images. À chaque clic, les images se repositionnement dynamiquement dans l'espace 3D en exerçant une rotation sur l'axe Y (voir Figure 9.10).

Figure 9.10

Aperçu
du document
publié.



Dans la scène principale du document, une série de MovieClip est répartie vers les calques (voir Figure 9.11).

Chaque symbole comporte une image doublée verticalement pour simuler un reflet. Le premier d'entre eux est situé précisément à 700 pixels à droite. Dans le code, nous utilisons un pas d'incrémentation de 300 pixels pour définir le repositionnement en X de ces images. La scène, elle, mesure 800 pixels et son centre est donc situé à 400 pixels (voir Figure 9.12).

En plaçant la première image à une valeur multiple du pas d'incrémentation (400 pixels + 300 pixels), nous permettons à l'ensemble des images d'apparaître plus tard, au centre de la scène, frontalement.

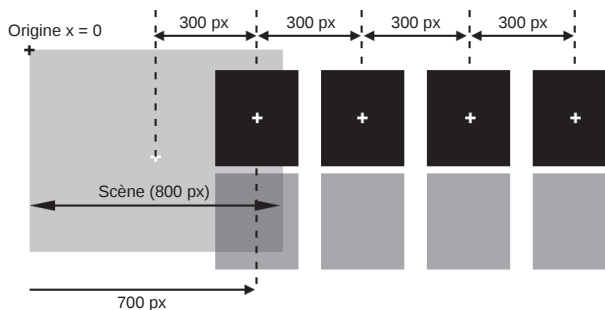
Les images sont par ailleurs collées les unes aux autres, et pourraient même se chevaucher, car la rotation Y que nous ajoutons sur chaque image en programmation, réduit leur espace nécessaire en largeur. De ce fait, une marge due à l'inclinaison apparaîtra lors de la publication.

Figure 9.11

Aperçu de la scène principale.

**Figure 9.12**

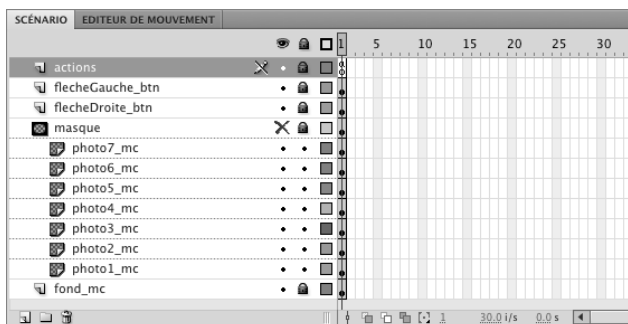
Calcul du positionnement des images.



Les flèches gauche et droite situées de part et d'autre du document risquent d'être cachées par le défilement des images. Pour contourner ce problème, dans le scénario, un masque qui affecte toutes les images a été appliqué et réduit la partie visible de la zone de défilement (voir Figure 9.13).

Figure 9.13

Aperçu du scénario de la scène principale.



À l'intérieur de chaque MovieClip d'image, le visuel est en fait contenu dans un second symbole. Ce dernier est dupliqué et placé symétriquement en miroir bas de façon à obtenir un effet de reflet. Sur ce deuxième symbole, nous avons appliqué un alpha à 40 %.

Tous les symboles de la scène possèdent chacun leur propre nom d'occurrence. Le code placé dans la fenêtre du calque Actions affiche ceci :

```
//----- initialisation
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

//----- actions

// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    TweenMax.to(photo1_mc, 2, {x:photo1_mc.x-300, delay:0, ease:Back.easeInOut});
    TweenMax.to(photo2_mc, 2, {x:photo2_mc.x-300, delay:0, ease:Back.easeInOut});
    TweenMax.to(photo3_mc, 2, {x:photo3_mc.x-300, delay:0, ease:Back.easeInOut});
    TweenMax.to(photo4_mc, 2, {x:photo4_mc.x-300, delay:0, ease:Back.easeInOut});
    TweenMax.to(photo5_mc, 2, {x:photo5_mc.x-300, delay:0, ease:Back.easeInOut});
    TweenMax.to(photo6_mc, 2, {x:photo6_mc.x-300, delay:0, ease:Back.easeInOut});
    TweenMax.to(photo7_mc, 2, {x:photo7_mc.x-300, delay:0, ease:Back.easeInOut});
}

// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
    TweenMax.to(photo1_mc, 2, {x:photo1_mc.x+300,delay:0, ease:Back.easeInOut});
    TweenMax.to(photo2_mc, 2, {x:photo2_mc.x+300,delay:0, ease:Back.easeInOut});
    TweenMax.to(photo3_mc, 2, {x:photo3_mc.x+300,delay:0, ease:Back.easeInOut});
    TweenMax.to(photo4_mc, 2, {x:photo4_mc.x+300,delay:0, ease:Back.easeInOut});
    TweenMax.to(photo5_mc, 2, {x:photo5_mc.x+300,delay:0, ease:Back.easeInOut});
    TweenMax.to(photo6_mc, 2, {x:photo6_mc.x+300,delay:0, ease:Back.easeInOut});
    TweenMax.to(photo7_mc, 2, {x:photo7_mc.x+300,delay:0, ease:Back.easeInOut});
}

// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;

addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
    if (photo1_mc.x<largeurScene+photo1_mc.width && photo1_mc.x>-(photo1_mc.width)) {
        photo1_mc.rotationY=Math.ceil((photo1_mc.x-moitieScene)/4);
    }
    if (photo2_mc.x<largeurScene+photo2_mc.width && photo2_mc.x>-(photo2_mc.width)) {
        photo2_mc.rotationY=Math.ceil((photo2_mc.x-moitieScene)/4);
    }
}
```

```

    if (photo3_mc.x<largeurScene+photo3_mc.width && photo3_mc.x>-(photo3_mc.width)) {
        photo3_mc.rotationY=Math.ceil((photo3_mc.x-moitieScene)/4);
    }
    if (photo4_mc.x<largeurScene+photo4_mc.width && photo4_mc.x>-(photo4_mc.width)) {
        photo4_mc.rotationY=Math.ceil((photo4_mc.x-moitieScene)/4);
    }
    if (photo5_mc.x<largeurScene+photo5_mc.width && photo5_mc.x>-(photo5_mc.width)) {
        photo5_mc.rotationY=Math.ceil((photo5_mc.x-moitieScene)/4);
    }
    if (photo6_mc.x<largeurScene+photo6_mc.width && photo6_mc.x>-(photo6_mc.width)) {
        photo6_mc.rotationY=Math.ceil((photo6_mc.x-moitieScene)/4);
    }
    if (photo7_mc.x<largeurScene+photo7_mc.width && photo7_mc.x>-(photo7_mc.width)) {
        photo7_mc.rotationY=Math.ceil((photo7_mc.x-moitieScene)/4);
    }
}

```

Ce code, conçu pour une gestion simple des contenus à partir d'objets placés directement dans le scénario, est structuré en trois parties. La première partie appelle les classes utilisées pour les transitions TweenMax. La deuxième partie gère le défilement horizontal des vignettes. La troisième partie assure la rotation dans l'espace 3D selon la position courante de chaque image dans la scène.

Le déplacement des images en X, engagé avec les flèches, utilise des interpolations de type TweenMax :

```

// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    TweenMax.to(photo1_mc, 2, {x:photo1_mc.x-300, delay:0, ease:Back.easeInOut});
}

```

Le principe est décliné pour chaque image de la scène et sur les deux flèches. Nous inversons simplement les valeurs de positionnement de +300 à -300.

Ensuite, nous organisons le système de rotation automatique avec un gestionnaire de type ENTER_FRAME, afin que l'événement soit recalculé perpétuellement :

```

// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;

addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
    if (photo1_mc.x<largeurScene+photo1_mc.width && photo1_mc.x>-(photo1_mc.width)) {
        photo1_mc.rotationY=Math.ceil((photo1_mc.x-moitieScene)/4);
    }
}

```

D'abord, nous initialisons deux valeurs. La première enregistre la largeur de la scène (stage.stageWidth). La seconde enregistre la position du centre de la scène, en divisant la première par deux (largeurScene/2). Ces valeurs sont utilisées dans la fonction

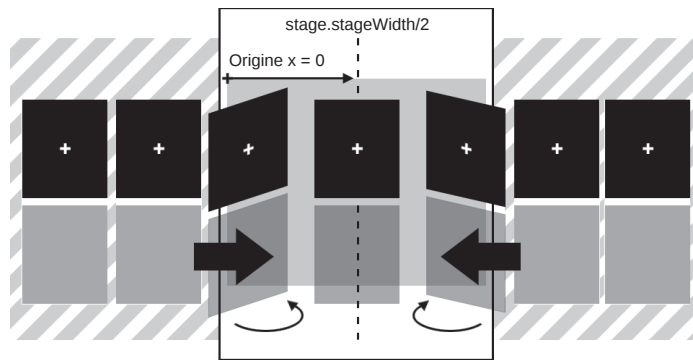
position3DAuto. Dans le bloc d'instructions de la fonction, pour chaque image, une condition est vérifiée :

```
if (photo1_mc.x < largeurScene + photo1_mc.width && photo1_mc.x > -(photo1_mc.width)) {
    photo1_mc.rotationY = Math.ceil((photo1_mc.x - moitieScene) / 4);
}
```

Si la position courante de l'image est inférieure à la largeur de la scène et sa largeur cumulée – donc si l'image entre dans le champ par la droite –, alors le programme calcule sa rotation en Y. Simultanément, une seconde condition est vérifiée. Elle se distingue de la première par les deux esperluettes (&&) : si, la position courante de l'image est également supérieure à sa largeur négative, c'est-à-dire, dès que l'image entre dans le champ par la gauche, alors, la même instruction que pour la première condition est exécutée (voir Figure 9.14).

Figure 9.14

Mécanisme de rotation en Y.



L'instruction désignée est la suivante :

```
photo1_mc.rotationY = Math.ceil((photo1_mc.x - moitieScene) / 4);
```

Nous définissons la rotation Y de l'objet en fonction de sa position courante en X. Plus précisément, nous reprenons la position en X de l'objet moins la demie largeur de la scène, de sorte que la rotation soit neutralisée au centre de la scène, et non à l'extrémité gauche où x vaut 0. Pour définir un angle de rotation entier, nous arrondissons la valeur grâce à l'utilisation de la méthode `Math.ceil()`.

Ce principe est appliqué à chacune des images présentes dans la scène.

Version dynamique

Afin de vous permettre d'exploiter plus facilement le carrousel avec des contenus externalisés, nous avons optimisé la gestion de ce programme en utilisant une structure composée à partir d'une boucle `for` et une autre avec une structure `for each... in`.



Exemples > ch9_3DNative_2b.fla
Exemples > ch9_3DNative_2c.fla

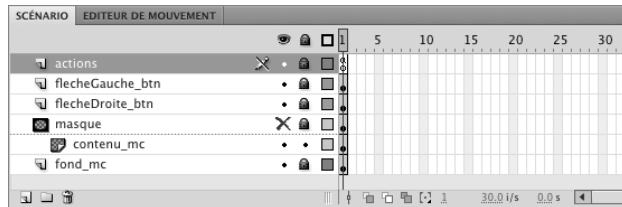
Dans le premier document, nous retrouvons quasiment la même structure que dans le fichier initial, à la différence que l'ensemble des vignettes est désormais rassemblé dans un conteneur

de type MovieClip que vous pourrez substituer éventuellement par un conteneur généré dynamiquement.

Cette modification structurelle, en ajoutant un conteneur, permet d'appliquer dynamiquement le même comportement à l'ensemble des éléments qu'il contient. Dans le scénario de la scène principale, le symbole contenu_mc rassemble tous les éléments (voir Figure 9.15).

Figure 9.15

Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var positionPhoto1:Number=(contenu_mc.getChildAt(0)).x;
var pasIncrementation:Number=300;

//----- actions

// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    for (var i:Number=0;i<contenu_mc.numChildren;i++) {
        TweenMax.to(
            (contenu_mc.getChildAt(i)), 2,
            {
                x:(contenu_mc.getChildAt(i)).x-pasIncrementation,
                delay:0,
                ease:Back.easeInOut
            }
        );
    }
}

// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
    for (var j:Number=0;j<contenu_mc.numChildren;j++) {
        TweenMax.to(
            (contenu_mc.getChildAt(j)), 2,
            {
                x:(contenu_mc.getChildAt(j)).x+pasIncrementation,
                delay:0,
                ease:Back.easeInOut
            }
        );
    }
}
```

```

    );
  }
}

// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;

contenu_mc.addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
    for (var k:Number=0;k<contenu_mc.numChildren;k++) {
        if
➤ ((contenu_mc.getChildAt(k)).x<largeurScene+(contenu_mc.getChildAt(k)).width-
➤ positionPhoto1 && (contenu_mc.getChildAt(k)).x>-largeurScene) {

        (contenu_mc.getChildAt(k)).rotationY=Math.ceil(((contenu_mc.getChildAt(k)).
➤ x-moitieScene-positionPhoto1)/4);
        }
    }
}

// ciblage dynamique de contenu
contenu_mc.addEventListener(MouseEvent.CLICK,actionPhotos);
function actionPhotos (evt:MouseEvent) {
    trace(evt.target.name);
    TweenMax.to(evt.target,2, {rotationZ:evt.target.rotationZ+360, delay:0,
    ease:Back.easeInOut});
}

```

Dans ce programme, nous initialisons d'abord deux valeurs que nous utiliserons dans le script. La première définit la position actuelle du premier symbole image dans `contenu_mc`. La méthode `getChildAt(0)` permet de sélectionner l'élément qui apparaît en premier dans la liste d'affichage de ce conteneur. Vous pouvez en vérifier le nom en ajoutant la propriété `name` à une action `trace` :

```
trace(contenu_mc.getChildAt(0).name); //photo1_mc
```

Dans les actions, le code est aéré sur plusieurs lignes afin de mieux distinguer chacune des propriétés utilisées dans l'interpolation `TweenMax`. Cela n'a pas d'incidence sur l'exécution du programme.

Nous remplaçons ici la répétition des instructions par une boucle `for` qui ne conserve qu'une seule ligne d'instruction. Il existe autant d'itérations que d'images. C'est la boucle qui, en fonction de ses paramètres, va multiplier et adapter les lignes de commande pour chaque contenu :

```

for (var i:Number=0;i<contenu_mc.numChildren;i++) {
    TweenMax.to(
        (contenu_mc.getChildAt(i)), 2, {x:(contenu_mc.getChildAt(i)).
        x-pasIncrementation, delay:0, ease:Back.easeInOut });
}

```

La valeur qui permet d'adapter l'instruction est véhiculée par `i`. Nous indiquons, au travers du constructeur `for`, d'initialiser `i` à 0. Puis, tant que cette valeur reste inférieure au nombre d'enfants (symboles ou objets) disponibles dans `contenu_mc`, nous incrémentons cette valeur. Ainsi, la boucle répète l'ensemble des instructions qu'elle contient, entre ses accolades,

autant de fois qu'il y aura d'autres images. À chaque itération, la valeur de `i` est augmentée (`i++`) si bien que les instructions qui utilisent cette valeur comme paramètre peuvent maintenant atteindre tour à tour chaque objet.

Dans l'interpolation `TweenMax`, nous ciblons chaque objet individuellement à l'aide de la méthode `getChildAt()` (avec `(contenu_mc.getChildAt(i))`), déjà invoquée pour la variable, plus haut. Cependant, il n'est pas possible d'appliquer directement la propriété `x` à la méthode `getChildAt()`. Des parenthèses sont donc ajoutées ici, pour contenir la première instruction qui cible d'abord l'objet, avant, seulement, de pouvoir modifier une des propriétés de cet objet. Cette méthode est déclinée pour l'ensemble des actions du programme.

En fin de code, une dernière action est ajoutée au conteneur principal. Elle permet de cibler individuellement chacun des éléments de ce conteneur en usant de la propriété `target` et de lui appliquer une rotation sur l'axe `Z` :

```
// ciblage dynamique de contenu
contenu_mc.addEventListener(MouseEvent.CLICK,actionPhotos);
function actionPhotos (evt:MouseEvent) {
    trace(evt.target.name);
    TweenMax.to(evt.target,2, {rotationZ:evt.target.rotationZ+360, delay:0,
    ➤ ease:Back.easeInOut});
}
```



Pour en savoir plus sur le développement dynamique d'interfaces, reportez-vous au manuel d'Anne Tasso (*Apprendre à programmer en ActionScript 3*, éd. Eyrolles).

Une déclinaison de ce document est proposée, avec la structure `for each... in`, dans le fichier "ch9_3DNative_2c fla". Dans ce document, nous avons remplacé la boucle `for` avec une boucle de type `for each... in`. Ce type de boucle possède la particularité de désigner l'ensemble des objets d'un conteneur simultanément. Ce qui évite d'avoir à les cibler individuellement. C'est donc encore plus optimisé. Cela donne :

```
// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
    for each (var mc:MovieClip in contenu_mc){
        TweenMax.to(mc, 2,{x:mc.x+pasIncrementation, delay:0, ease:Back.easeInOut });
    }
}

// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
    for each (var mc:MovieClip in contenu_mc){
        TweenMax.to(mc, 2,{x:mc.x-pasIncrementation, delay:0, ease:Back.easeInOut });
    }
}

// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;
```

```

contenu_mc.addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
    for each (var mc:MovieClip in contenu_mc){
        if (mc.x<largeurScene+mc.width-positionPhoto1 && mc.x>-largeurScene) {
            mc.rotationY=Math.ceil((mc.x-moitieScene-positionPhoto1)/4);
        }
    }
}

```

À retenir

- Pour centrer dans la scène les effets ou les objets, nous intégrons, dans le calcul des valeurs, la position du centre de la scène avec `stage.stageWidth/2`.
- La boucle `for each...` in permet de cibler directement l'ensemble des objets d'un conteneur.

Mur d'images 3D

Un mur d'images est une mosaïque sur laquelle on clique pour afficher une image en détail. Lorsque le pointeur survole l'écran, le mur d'images oscille proportionnellement dans un sens ou dans l'autre, verticalement et horizontalement.

Dans cet exemple, nous utilisons des symboles placés sur la scène dont nous contrôlons l'inclinaison en fonction de la position du pointeur à l'écran. En balayant la scène, le mur d'image bouge. Nous ajoutons une interactivité sur les images à l'aide de la propriété `target`, pour, par exemple, zoomer individuellement sur chaque image et matérialiser un effet de "rollOver" avec un filtre dynamique (voir Figures 9.16 à 9.18).

Figure 9.16

Aperçu du document à la publication, pointeur sur le côté.

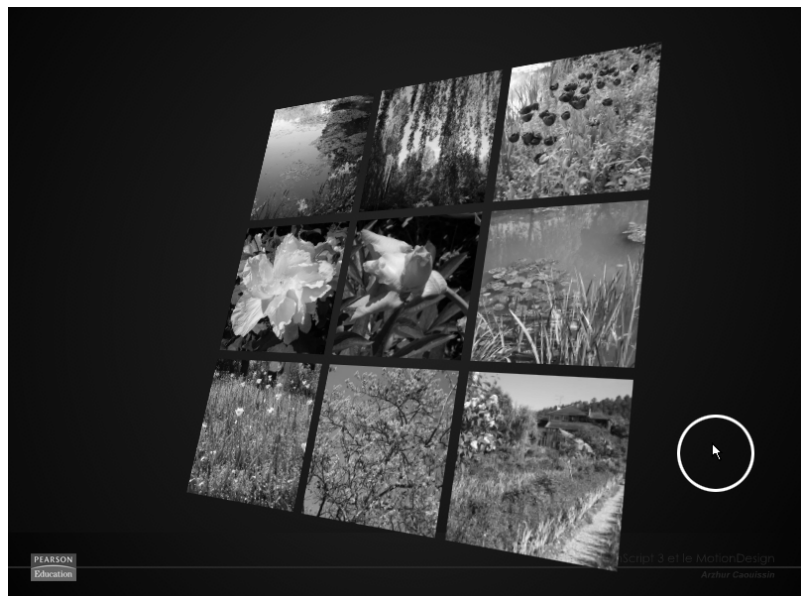


Figure 9.17

Aperçu du document à la publication, image survolée.

**Figure 9.18**

Aperçu du document à la publication, image cliquée et zoomée.



Exemples > ch9_3DNative_3 fla

Le document "ch9_3DNative_3 fla" affiche une mosaïque nommée contenu_mc, à l'intérieur de laquelle sont répartis différents MovieClip. Chacun de ces MovieClip contient sa propre image et possède un nom d'occurrence (voir Figure 9.19).

Figure 9.19

Aperçu de la scène principale.



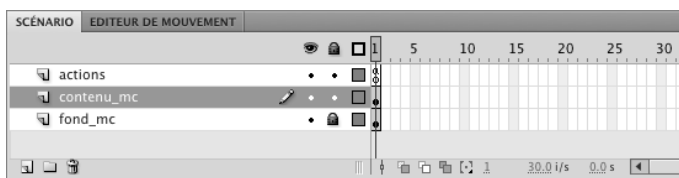
Chaque symbole est déjà réduit à 50 % de ses dimensions originales. Cela permet d'activer une fonction de zoom sans perdre sur la netteté de l'image. Le zoom leur affectera seulement leur échelle normale.

Pour améliorer également le rendu des images, dans un contexte en perpétuel mouvement, nous avons activé, depuis la fenêtre Bibliothèque, dans les propriétés de chaque image (clic-droit > Propriétés), l'option Autoriser le lissage.

Le scénario ne laisse apparaître que le symbole contenu_mc (voir Figure 9.20).

Figure 9.20

Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//----- initialisation
import flash.filters.*;
var haloIn:GlowFilter=new GlowFilter(0xffffffff, 1, 2, 2, 3, 255, false, false);
var haloOut:GlowFilter=new GlowFilter(0xffffffff, 1, 0, 0, 3, 255, false, false);

import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var demieScene:Number=stage.stageWidth/2;
var hauteurScene:Number=stage.stageHeight/2;
var Tween3D:TweenMax;
```

```

var nomImageActive:String;

var positionInitX:Array=new Array();
var positionInitY:Array=new Array();
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
    positionInitX.push(contenu_mc.getChildAt(i).x)
    positionInitY.push(contenu_mc.getChildAt(i).y)
}

//----- actions

// mouvement 3D
contenu_mc.addEventListener(Event.ENTER_FRAME, murImages);
function murImages (evt:Event) {
    contenu_mc.rotationY=(mouseX-demieScene)*0.1;
    contenu_mc.rotationX=(mouseY-hauteurScene)*-0.1;
}

// interactivit  zoom
contenu_mc.addEventListener(MouseEvent.CLICK, zoom);
function zoom (evt:MouseEvent) {
    Tween3D=TweenMax.to(contenu_mc.photo1_mc, 0.3, {z:0, x:positionInitX[0],
    ➤ y:positionInitY[0], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo2_mc, 0.3, {z:0, x:positionInitX[1],
    ➤ y:positionInitY[1], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo3_mc, 0.3, {z:0, x:positionInitX[2],
    ➤ y:positionInitY[2], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo4_mc, 0.3, {z:0, x:positionInitX[3],
    ➤ y:positionInitY[3], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo5_mc, 0.3, {z:0, x:positionInitX[4],
    ➤ y:positionInitY[4], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo6_mc, 0.3, {z:0, x:positionInitX[5],
    ➤ y:positionInitY[5], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo7_mc, 0.3, {z:0, x:positionInitX[6],
    ➤ y:positionInitY[6], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo8_mc, 0.3, {z:0, x:positionInitX[7],
    ➤ y:positionInitY[7], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo9_mc, 0.3, {z:0, x:positionInitX[8],
    ➤ y:positionInitY[8], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    //
    Tween3D.addEventListener(TweenEvent.COMPLETE, restaureFini);
    function restaureFini (yo:TweenEvent) {
        contenu_mc.addChild(MovieClip(evt.target));
    }
    if (nomImageActive!=evt.target.name) {
        nomImageActive=evt.target.name
        //
        TweenMax.to(evt.target,2, {z:-100, x:0, y:0, scaleX:1.2, scaleY:1.2,
        ➤ delay:0.3, ease:Elastic.easeOut});
    } else {
        nomImageActive="";
    }
}

// rollOvers
contenu_mc.addEventListener(MouseEvent.MOUSE_OVER, over);

```



```

function over (evt:MouseEvent) {
    evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    evt.target.filters=[haloOut];
}

```

Le code est composé de quatre parties. L'initialisation, les actions de positionnement du mur dans l'espace 3D, la gestion de l'interactivité (zoom) sur les images et la gestion de l'effet rollOver.

Avant de détailler chaque étape, nous devons comprendre le processus employé. En publiant le document, d'abord, l'ensemble nommé `conteneur_mc` oscille en fonction de la position du pointeur dans la scène. Puis, au passage de la souris sur les symboles qu'il contient, ceux-ci réagissent avec un filtre dynamique. En cliquant sur un symbole, celui-ci s'agrandit. En cliquant à nouveau dessus, il revient à sa position normale. Mais, si nous cliquons directement sur un autre symbole, simultanément, la nouvelle image s'agrandit, tandis que la précédente revient en position initiale.

Dans la première partie, nous commençons par importer les classes requises, dont `filters`, qui permettent, plus bas, de gérer l'effet rollOver (voir Chapitre 2). Nous commençons par instancier les deux effets, l'effet survolé et l'effet inactif. Lorsque le pointeur survolera l'image, le premier sera activé. Le second sera appelé en sortant de l'image :

```

//----- initialisation
import flash.filters.*;
var haloIn:GlowFilter=new GlowFilter(0xffffffff, 1, 2, 2, 3, 255, false, false);
var haloOut:GlowFilter=new GlowFilter(0xffffffff, 1, 0, 0, 3, 255, false, false);

```

Nous importons ensuite les classes des transitions `TweenMax` :

```

import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

```

Puis, nous définissons quelques valeurs :

```

var demieScene:Number=stage.stageWidth/2;
var hauteurScene:Number=stage.stageHeight/2;
var Tween3D:TweenMax;
var nomImageActive:String;

```

Les deux premières valeurs servent à définir les rotations par rapport au centre de la scène, sur le même principe que celui énoncé dans la section précédente. Ensuite, nous typons une interpolation `TweenMax` afin de pouvoir enchaîner des actions à l'issue d'une interpolation.

Enfin, nous créons une variable chaîne de caractères afin de capturer, à chaque clic, le nom de l'image cliquée (`nomImageActive`), de manière à différencier son comportement, selon qu'elle ait déjà été zoomée ou non.

Nous définissons ensuite deux tableaux qui enregistrent respectivement les positions de départ des images en X et Y. Nous utiliserons, plus loin dans le code, ces valeurs afin de restaurer les images à leurs positions initiales :

```

var positionInitX:Array=new Array();
var positionInitY:Array=new Array();

```

```

for (var i:Number=0; i<contenu_mc.numChildren; i++) {
    positionInitX.push(contenu_mc.getChildAt(i).x)
    positionInitY.push(contenu_mc.getChildAt(i).y)
}

```

La création de chaque tableau est lancée avec la classe Array. Une boucle for ajoute, grâce à la méthode `push()`, la position courante, en X et en Y, de chaque objet en fonction de son ordre d’affichage (`contenu_mc.getChildAt(i)`). La boucle exécute autant d’itération qu’il y a d’enfant dans le symbole `contenu_mc` (`contenu_mc.numChildren`).

Dans les actions, nous commençons par définir le mouvement global du conteneur d’images :

```

//----- actions

// mouvement 3D
contenu_mc.addEventListener(Event.ENTER_FRAME, murImages);
function murImages (evt:Event) {
    contenu_mc.rotationY=(mouseX-demieScene)*0.1;
    contenu_mc.rotationX=(mouseY-hauteurScene)*-0.1;
}

```

Un écouteur est attaché au contenu et fait pivoter, en continu, l’objet sur l’axe Y (`rotationY`) et sur l’axe X (`rotationX`). La rotation sur l’axe Y est déterminée en fonction de la position du pointeur sur X. Celle de l’axe X, en fonction de la position du pointeur sur Y. Mais, comme vu précédemment, nous retranchons la demi-largeur et hauteur de la scène, afin de neutraliser les rotations lorsque le pointeur survole le milieu de l’écran (quand la position vaut 0, la `rotationX` et `rotationY` vaut 0).

D’autre part, la position du pointeur définie en X et en Y se mesure en centaines de pixels. La rotation, elle, se mesure en dizaines de degrés. Nous réduisons donc la valeur obtenue en la multipliant par 0.1 (équivalent à diviser par 10), ceci afin d’éviter un mouvement trop prononcé.

Nous ajustons également la polarité du mouvement, avec un signe moins, selon que le sens que l’on veut affecter à l’inclinaison. Le signe moins inverse l’inclinaison par rapport à la position du pointeur. L’absence de signe conserve une inclinaison qui accompagne le mouvement du pointeur.

À la suite, nous créons l’interactivité relative à la fonction zoom, appliquée à chaque image :

```

// interactivité zoom
contenu_mc.addEventListener(MouseEvent.CLICK, zoom);
function zoom (evt:MouseEvent) {
    Tween3D=TweenMax.to(contenu_mc.photo1_mc, 0.3, {z:0, x:positionInitX[0],
    ➤ y:positionInitY[0], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo2_mc, 0.3, {z:0, x:positionInitX[1],
    ➤ y:positionInitY[1], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo3_mc, 0.3, {z:0, x:positionInitX[2],
    ➤ y:positionInitY[2], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo4_mc, 0.3, {z:0, x:positionInitX[3],
    ➤ y:positionInitY[3], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.photo5_mc, 0.3, {z:0, x:positionInitX[4],
    ➤ y:positionInitY[4], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
}

```

```

TweenMax.to(contenu_mc.photo6_mc, 0.3, {z:0, x:positionInitX[5],
➤ y:positionInitY[5], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
TweenMax.to(contenu_mc.photo7_mc, 0.3, {z:0, x:positionInitX[6],
➤ y:positionInitY[6], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
TweenMax.to(contenu_mc.photo8_mc, 0.3, {z:0, x:positionInitX[7],
➤ y:positionInitY[7], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
TweenMax.to(contenu_mc.photo9_mc, 0.3, {z:0, x:positionInitX[8],
➤ y:positionInitY[8], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
//
Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFini);
function restaureFini (yo:TweenEvent) {
    contenu_mc.addChild(MovieClip(evt.target));
}
// autres actions
}

```

Dans un premier temps, nous appliquons un écouteur qui détecte tout clic capturé par l'objet contenu_mc. Le principe de propagation événementielle utilisé par AS3 permet aux objets contenus par un conteneur d'intercepter un événement appliqué à celui-ci. Nous utilisons pour cela la propriété target et nous désignons un écouteur global pour l'ensemble des symboles qu'il contient.

En cliquant sur l'un des symboles, une interpolation TweenMax restaure d'abord la position initiale de ceux qui ont éventuellement déjà été zoomés. Cette interpolation est très courte (0.3 secondes) afin de ne pas créer une attente trop longue lorsqu'aucune image n'a encore été activée (au premier clic). Les propriétés animées sont l'index z, qui définit la position de l'image en profondeur. De valeur 0, elle désigne une position normale.

Nous restaurons aussi la position courante du symbole en X et Y, ainsi que son échelle scaleX et scaleY, à 50 % (0.5).

L'interpolation créée pour le premier symbole est déclinée pour l'ensemble des éléments à animer, soit 9 fois. Pour chacun d'entre eux, nous modifions le nom de l'objet à manipuler et sa position en X et Y. Les valeurs utilisées sont celles stockées initialement dans les tableaux. Pour lire ces valeurs, nous reprenons le nom de chaque tableau en ajoutant deux crochets, avec, en paramètre, la valeur correspondant au symbole ciblé (positionInitX[0] et positionInitY[0]).

La première d'entre elles est identifiée avec un nom de variable tween3D. Cette instantiation nous permet d'ajouter une fonction une fois l'interpolation de restauration achevée.

La fonction restaureFini, qui est appelée, est placée dans la fonction zoom, car elle fait directement référence à l'objet cliqué avec target par le biais de la variable evt, utilisée en paramètre de la fonction zoom. Si la fonction était placée en dehors de zoom, l'objet ne serait plus identifié. Si nous avions employé le même nom de variable en paramètre de la fonction restaureFini, nous aurions aussi généré une erreur, car le lecteur n'aurait pas su à quelle classe il devait se référer, MouseEvent ou bien TweenEvent, pour recevoir les informations. C'est la raison pour laquelle cette fonction emploie également un terme différencié du premier et sans valeur spécifique (yo).

La fonction restaureFini commence donc par redistribuer au sommet de la liste d'affichage, l'objet cliqué. Cette action replace virtuellement l'objet sur le calque du dessus, dans

contenu_mc, en faisant retomber la pile d'objets restant, un niveau en dessous, jusqu'à l'emplacement alors resté vide. Ainsi, l'image zoomée apparaîtra au premier plan indépendamment des autres, et non en chevauchant les autres images.

La fonction se poursuit avec une structure conditionnelle :

```
Tween3D.addEventListener(TweenEvent.COMPLETE, restaureFini);
function restaureFini (yo:TweenEvent) {
    contenu_mc.addChild(MovieClip(evt.target));
}
if (nomImageActive!=evt.target.name) {
    nomImageActive=evt.target.name
    //
    TweenMax.to(evt.target, 2, {z:-100, x:0, y:0, scaleX:1.2, scaleY:1.2,
    ➡ delay:0.3, ease:Elastic.easeOut});
} else {
    nomImageActive="";
}
}
```

Si le nom du symbole cliqué (evt.target.name) est différent (!=) du nom enregistré dans la variable (evt.target.name) ou si l'image n'a pas déjà été cliquée, alors, une action est exécutée.

Cette action commence par mémoriser le nom de l'objet cliqué afin de pouvoir l'identifier comme actif lors du prochain clic.

La deuxième instruction est une interpolation. Elle cible l'objet cliqué (evt.target) et lui affecte un index de profondeur -100. L'objet est donc rapproché de l'écran. La position en X et Y est recentrée afin que l'image agrandie ne sorte pas de l'écran lorsqu'un symbole situé dans les coins est activé. Puis, en plus du changement d'index z, l'image est agrandie à 120 %. Comme nous avons activé le lissage des pixels, cette valeur reste tolérable et la déformation demeure encore imperceptible.

Si la condition else n'est pas vérifiée, nous purgeons le nom enregistré. Cela permet que lorsque l'utilisateur clique directement deux fois sur la même image, de l'ouvrir à nouveau, sans devoir cliquer au préalable sur une autre image. Tant que le nom de l'image n'est pas modifié, l'image déjà cliquée ne pourra pas en effet être rejouée, sauf si nous modifions ce nom en le substituant, par exemple, avec un texte vide ("").

Le programme s'achève avec la gestion des rollover :

```
// rollovers
contenu_mc.addEventListener(MouseEvent.CLICK, over);
function over (evt:MouseEvent) {
    evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.CLICK, out);
function out (evt:MouseEvent) {
    evt.target.filters=[haloOut];
}
```

En passant avec le pointeur sur les éléments du conteneur, le filtre dynamique `GlowFilter` (`filters=[haloIn]`) affiche un halo blanc autour de l'objet survolé (`evt.target`). De la même manière, en sortant le pointeur de l'objet survolé, le filtre est initialisé avec le second objet `GlowFilter` (`filters=[haloOut]`).

À retenir

- Pour regrouper le stockage de valeurs, nous pouvons utiliser un tableau grâce à la méthode `Array`. Puis les redistribuer en l'invoquant par son nom et en paramètre duquel nous spécifions l'objet stocké à lire.
- Afin de placer un symbole au premier plan, dans un espace 3D, nous devons le repositionner au sommet de la liste d'affichage, avec, par exemple, la méthode `addChild()`.
- Lorsque plusieurs fonctions sont imbriquées, il est souhaitable de distinguer les identifiants passés en paramètre, afin de distinguer les classes distribuées par les différents écouteurs.

Galerie vidéo 3D circulaire

Une galerie en 3D avec de la vidéo peut paraître similaire, d'un point de vue structurel, à une galerie 3D d'images. Mais, pour obtenir une image propre et un document optimisé, vous devez tenir compte de certaines distinctions :

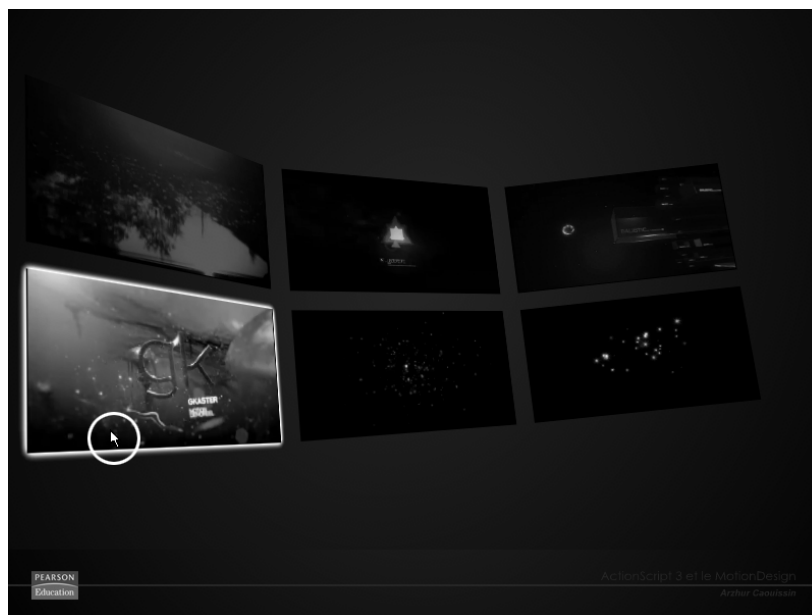
- L'image jouée en vidéo doit être dimensionnée en fonction de l'affichage final.
- Lors des transitions, pour préserver la qualité de l'image, c'est le composant vidéo qui doit être redimensionné et non son conteneur. C'est donc le conteneur qui évolue dans l'espace 3D et non la vidéo.
- Les vidéos doivent de préférence être arrêtées si aucune interaction de l'utilisateur n'est avérée. Ceci afin d'optimiser les ressources machines de l'utilisateur déjà sollicitées par la gestion de l'affichage 3D.

Dans cet exemple, nous présentons une galerie d'écrans vidéo disposés de manière circulaire. La galerie oscille selon la position du pointeur sur le même mécanisme que l'exemple précédent. En survolant les écrans, un effet de `rollOver` se produit, et, simultanément, la vidéo survolée démarre. En cliquant sur la vidéo, celle-ci est projetée frontalement avec un zoom spatial (échelle + `index z`) et l'oscillation est arrêtée. Lorsque l'utilisateur clique à nouveau sur la vidéo, celle-ci revient à sa position initiale (voir Figures 9.21 et 9.22).

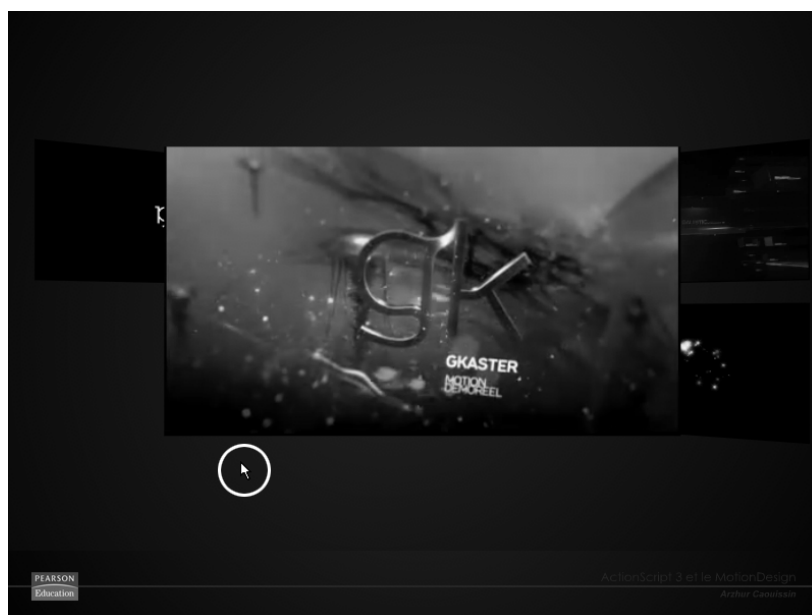
Dans cette section, nous proposons deux approches pour la construction de cette interface. La première, accessible facilement, met en forme le dispositif à partir d'instructions simples et répétées. Une seconde approche exploite à l'inverse une structure tabulaire pour véhiculer l'ensemble des propriétés des objets, afin de simplifier la gestion du code. Vous trouverez donc, nécessairement, la formule qui vous convient le mieux pour réaliser vos propres systèmes d'affichage en 3D.

Figure 9.21

Aperçu du document à la publication, vidéo survolée.

**Figure 9.22**

Aperçu du document à la publication, vidéo projetée frontalement.



La galerie simple

Dans cette version, nous initialisons les objets à mettre en forme à partir d'instructions répétées. Puis, nous ajoutons les comportements requis pour les animations et l'interactivité.



Exemples > ch9_3DNative_4 fla

Dans le document "ch9_3DNative_4 fla", sur la scène principale, le symbole contenu_mc affiche six occurrences du même MovieClip, mais de noms différents. Ce MovieClip contient un composant FLVPlayback nommé ecranVideo (voir Figure 9.23).

Figure 9.23

Aperçu de la scène principale.



Dans la fenêtre de scénario, au-dessus du calque fond_mc, on distingue le symbole contenu_mc (voir Figure 9.24).

Figure 9.24

Aperçu du scénario.



La fenêtre Actions affiche le code suivant :

```
//----- initialisation
import flash.filters.*;
```

```

var haloIn:GlowFilter=new GlowFilter(0xffffffff, 1, 2, 2, 3, 255, false, false);
var haloOut:GlowFilter=new GlowFilter(0xffffffff, 0, 2, 2, 3, 255, false, false);

import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var demieScene:Number=stage.stageWidth/2;
var hauteurScene:Number=stage.stageHeight/2;
var Tween3D:TweenMax;
var nomImageActive:String;

//----- configuration des écrans vidéo
contenu_mc.v1_mc.ecranVideo.source="video3D/3d-gKaster-C19.f4v";
contenu_mc.v2_mc.ecranVideo.source="video3D/3d-gKaster-amusement.f4v";
contenu_mc.v3_mc.ecranVideo.source="video3D/3d-gKaster-balistic.f4v";
contenu_mc.v4_mc.ecranVideo.source="video3D/3d-gKaster-demoreel.f4v";
contenu_mc.v5_mc.ecranVideo.source="video3D/3d-galaxieFull.f4v";
contenu_mc.v6_mc.ecranVideo.source="video3D/3d-particules.f4v";
//
contenu_mc.v1_mc.rotationY=-30;
contenu_mc.v2_mc.rotationY=0;
contenu_mc.v3_mc.rotationY=30;
contenu_mc.v4_mc.rotationY=-30;
contenu_mc.v5_mc.rotationY=0;
contenu_mc.v6_mc.rotationY=30;
//
contenu_mc.v1_mc.z=-25;
contenu_mc.v2_mc.z=40;
contenu_mc.v3_mc.z=-25;
contenu_mc.v4_mc.z=-25;
contenu_mc.v5_mc.z=40;
contenu_mc.v6_mc.z=-25;
//
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
    MovieClip(contenu_mc.getChildAt(i)).ecranVideo.stop();
    MovieClip(contenu_mc.getChildAt(i)).ecranVideo.scaleMode="exactFit";
    MovieClip(contenu_mc.getChildAt(i)).ecranVideo.autoPlay=false;
}

//----- actions
// mur d'image 3D
contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
function murImages (evt:Event) {
    contenu_mc.rotationY=(mouseX-demieScene)*0.1;
    contenu_mc.rotationX=(mouseY-hauteurScene)*-0.1;
}

// interactivité vidéo
contenu_mc.addEventListener(MouseEvent.CLICK,zoomVideo);
function zoomVideo (evt:MouseEvent) {
    //
    contenu_mc.removeEventListener(Event.ENTER_FRAME,murImages);
    TweenMax.to(contenu_mc, 3, {rotationX:0, rotationY:0, delay:0,
    ➤ ease:Elastic.easeInOut});
    //

```



```

Tween3D=TweenMax.to(contenu_mc.v1_mc, 0.3, {z:-25, x:-245, y:-72,
➤ rotationY:-30, delay:0, ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v2_mc, 0.3, {z:40, x:0, y:-72, delay:0, rotationY:0,
➤ ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v3_mc, 0.3, {z:-25, x:245, y:-72, delay:0,
➤ rotationY:30, ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v4_mc, 0.3, {z:-25, x:-245, y:72, delay:0,
➤ rotationY:-30, ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v5_mc, 0.3, {z:40, x:0, y:72, delay:0, rotationY:0,
➤ ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v6_mc, 0.3, {z:-25, x:245, y:72, delay:0,
➤ rotationY:30, ease:Strong.easeInOut});
//
TweenMax.to(contenu_mc.v1_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v2_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v3_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v4_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v5_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v6_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
//
Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFini);
function restaureFini (yo:TweenEvent) {
    contenu_mc.addChild(MovieClip(evt.target));
}
if (nomImageActive!=evt.target.name) {
    nomImageActive=evt.target.name
    //
    TweenMax.to(evt.target.ecranVideo,2, {width:440, height:246, x:-220,
➤ y:-123, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(evt.target, 2, {z:-100, x:0, y:0, rotationY:0, delay:0.3,
➤ ease:Elastic.easeOut});
} else {
    nomImageActive="";
    contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
}
}

// rollOvers
contenu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    evt.target.ecranVideo.play();
    evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    evt.target.ecranVideo.stop();
    evt.target.filters=[haloOut];
}

```

Le programme se décompose en cinq parties. La première importe les classes requises dont les filtres et les transitions TweenMax. La deuxième initialise les caractéristiques de chaque composant vidéo et le positionnement de son conteneur dans l'espace 3D. La troisième

active l'animation 3D du mur d'écrans. La quatrième lance l'interactivité avec les flux vidéo. La cinquième, enfin, ajoute l'effet `rollOver` sur chaque vidéo et active la lecture de cette vidéo.

Dans la deuxième partie, chaque composant est initialisé selon les paramètres suivants. L'ensemble de ces réglages est décliné pour chacune des vidéos. Voici les caractéristiques pour l'une d'entre elles :

```
//----- configuration des écrans vidéo
contenu_mc.v1_mc.ecranVideo.source="video3D/3d-gKaster-C19.f4v";
//
contenu_mc.v1_mc.rotationY=-30;
//
contenu_mc.v1_mc.z=-25;
```

Analysons le code :

- La propriété `source` désigne l'URL du fichier vidéo à lire.
- Les propriétés `rotationY` et `rotationZ` positionnent non pas le composant mais son conteneur, le `MovieClip`, dans l'espace 3D.

Plus loin, nous utilisons une boucle `for` afin de regrouper la gestion de l'ensemble des composants vidéo distribués dans chaque `MovieClip` :

```
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
    MovieClip(contenu_mc.getChildAt(i)).ecranVideo.stop();
    MovieClip(contenu_mc.getChildAt(i)).ecranVideo.scaleMode="exactFit";
    MovieClip(contenu_mc.getChildAt(i)).ecranVideo.autoPlay=false;
}
```

Analysons le code :

- La méthode `stop()`, une fois la source appelée, permet de préserver les ressources de l'utilisateur en empêchant les vidéos de se lire automatiquement.
- La propriété `scaleMode` indique si le composant doit être redimensionné en fonction des dimensions réelles de la vidéo. La valeur `exactFit` interdit le redimensionnement. Si la vidéo est plus grande ou de proportions différentes du composant, elle épousera quand même les dimensions du composant (voir Chapitre 8 pour le descriptif détaillé de ces paramètres).
- Enfin, la propriété `autoPlay` passée sur `false` empêche la vidéo de lire au démarrage (deux précautions valent mieux qu'une).

Plus loin, les actions affichent le contrôle du mouvement du mur d'écrans (voir section précédente). À la suite, apparaissent les contrôles d'interactivité avec les flux vidéos :

```
// interactivité vidéo
contenu_mc.addEventListener(MouseEvent.CLICK, zoomVideo);
function zoomVideo (evt:MouseEvent) {
    //
    contenu_mc.removeEventListener(Event.ENTER_FRAME, murImages);
    TweenMax.to(contenu_mc, 3, {rotationX:0, rotationY:0, delay:0,
        ➤ ease:Elastic.easeInOut});
}
```

Dans un premier temps, la fonction, appelée lorsque l'utilisateur active une vidéo, neutralise l'oscillation du mur (`removeEventListener`). Ainsi, la vidéo peut être repositionnée frontalement assez simplement et permettre une lecture confortable du contenu. Cette fonction sera réactivée lorsque toutes les vidéos seront restaurées à leur position initiale, plus loin dans le code.

Une deuxième instruction indique au symbole `contenu_mc`, dont nous venons d'interrompre le mouvement, de passer progressivement de sa position actuelle à une position frontale, pour laquelle tous les paramètres modifiés sont initialisés à la valeur 0 (`rotationX:0, rotationY:0`).

Ensuite, nous ajoutons les transitions pour chacun des composants et leurs conteneurs respectifs :

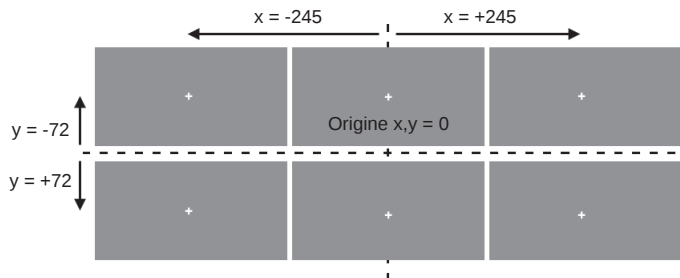
```
//
Tween3D=TweenMax.to(contenu_mc.v1_mc, 0.3, {z:-25, x:-245, y:-72,
➤ rotationY:-30, delay:0 ,ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v2_mc, 0.3, {z:40, x:0, y:-72, delay:0, rotationY:0,
➤ ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v3_mc, 0.3, {z:-25, x:245, y:-72, delay:0,
➤ rotationY:30, ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v4_mc, 0.3, {z:-25, x:-245, y:72, delay:0,
➤ rotationY:-30, ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v5_mc, 0.3, {z:40, x:0, y:72, delay:0, rotationY:0,
➤ ease:Strong.easeInOut});
TweenMax.to(contenu_mc.v6_mc, 0.3, {z:-25, x:245, y:72, delay:0,
➤ rotationY:30, ease:Strong.easeInOut});
//
TweenMax.to(contenu_mc.v1_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v2_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v3_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v4_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v5_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(contenu_mc.v6_mc.ecranVideo, 2, {width:220, height:123, x:-110,
➤ y:-62, delay:0.3, ease:Elastic.easeOut});
//
```

Vous relevez que nous animons distinctement le conteneur et le composant, avec différentes propriétés.

Si nous avons effectivement animé uniquement le conteneur, en lui appliquant les propriétés de redimensionnement qui nous permettent d'agrandir l'image, celle-ci aurait été considérablement détériorée, même avec un lissage. Or, la vidéo s'adapte par rapport aux dimensions du composant. Il convient donc de redimensionner le composant et non le conteneur. C'est la raison pour laquelle nous obtenons deux séries d'interpolations, affectant chacune tous les conteneurs (`contenu_mc.v1_mc`) – voir Figure 9.25 – puis tous les composants (`contenu_mc.v1_mc.ecranVideo`).

Figure 9.25

Calcul du positionnement des écrans, dans le conteneur contenu_mc.



Pour compenser néanmoins le décalage que peut provoquer le redimensionnement du composant indépendamment de son conteneur, nous rappelons les propriétés *x* et *y* de chaque objet pour recentrer l'ensemble à chaque modification d'échelle. Le changement de taille des éléments conduit en effet à les décaler de leur position d'origine. Il faut donc aussi animer et restaurer les propriétés *x* et *y* de chacun d'entre eux.

Pour permettre d'enchaîner ces transitions avec d'autres actions, l'une d'entre elles possède un identifiant *tween3D*.

Plus loin, nous ajoutons un écouteur à l'objet *tween3D*, comme vu à la section précédente. Nous enchaînons ici avec la condition suivante :

```
//
Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFin);
function restaureFin (yo:TweenEvent) {
    contenu_mc.addChild(MovieClip(evt.target));
}
if (nomImageActive!=evt.target.name) {
    nomImageActive=evt.target.name
    //
    TweenMax.to(evt.target.ecranVideo,2, {width:440, height:246, x:-220,
    ➤ y:-123, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(evt.target, 2, {z:-100, x:0, y:0, rotationY:0, delay:0.3,
    ➤ ease:Elastic.easeOut});
} else {
    nomImageActive=" ";
    contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
}
}
```

Nous spécifions d'abord de placer le conteneur cliqué au premier plan pour éviter les chevauchements lors du redimensionnement :

```
contenu_mc.addChild(MovieClip(evt.target));
```

Nous vérifions ensuite la condition selon laquelle l'objet cliqué n'est pas déjà zoomé, selon le même principe que celui utilisé dans la section précédente.

Les transitions affectent les deux objets (*evt.target.ecranVideo* et *evt.target*).

Le composant vidéo activé est agrandi selon les dimensions exactes observées durant l'encodage. Nous avons relevé les valeurs 440 × 246 pixels. Nous les appliquons donc aux propriétés *width* et *height* du composant vidéo. Le flux vidéo, projeté au premier plan,

s'adaptera ainsi en pleine résolution à la nouvelle dimension du composant. Le conteneur de la vidéo activée (`evt.target`) est placé, quant à lui, frontalement. Si la condition n'est pas vérifiée, donc, si aucune autre vidéo ne prend la place centrale, nous réactivons le mur d'images (`addEventListener`).

Le programme se termine avec la gestion des `rollOver` :

```
// rollOver
contenu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    evt.target.ecranVideo.play();
    evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    evt.target.ecranVideo.stop();
    evt.target.filters=[haloOut];
}
```

Lorsque l'utilisateur survole un conteneur, la vidéo qu'il véhicule est jouée (`evt.target.ecranVideo.play()`). Le filtre du halo blanc est exécuté (`evt.target.filters=[haloIn]`).

Les mêmes instructions, mais neutralisées, sont exécutées lorsque le pointeur quitte la surface de l'objet (`MOUSE_OUT`).

La galerie optimisée

Dans cette version, nous optimisons la gestion des propriétés de chaque objet avant de réaliser les animations et développer l'interactivité. Nous distribuons ensuite, sous la forme de commandes compactes, chacune des instructions déjà étudiées dans la version précédente.



Exemples > `ch9_3DNative_4b fla`

Le document possède une structure similaire au précédent. La fenêtre Actions affiche le code suivant :

```
//----- initialisation
import flash.filters.*;
var haloIn:GlowFilter=new GlowFilter(0xffffffff, 1, 2, 2, 3, 255, false, false);
var haloOut:GlowFilter=new GlowFilter(0xffffffff, 0, 2, 2, 3, 255, false, false);

import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var demieScene:Number=stage.stageWidth/2;
var hauteurScene:Number=stage.stageHeight/2;
var Tween3D:TweenMax;
var nomImageActive:String;
```

```
//----- configuration des écrans vidéo
var datas:Array = [
    {source:"video3D/3d-gKaster-C19.f4v",rotationY:-30,x:-245,y:-72,z:-25},
    {source:"video3D/3d-gKaster-amusement.f4v",rotationY:0,x:0,y:-72,z:40},
    {source:"video3D/3d-gKaster-balistic.f4v",rotationY:30,x:245,y:-72,z:-25},
    {source:"video3D/3d-gKaster-demoreel.f4v",rotationY:-30,x:-245,y:72,z:-25},
    {source:"video3D/3d-galaxieFull.f4v",rotationY:0,x:0,y:72,z:40},
    {source:"video3D/3d-particules.f4v",rotationY:30,x:245,y:72,z:-25}
];

for (var i:Number=0; i<contenu_mc.numChildren; i++) {
    contenu_mc["v"+i+"_mc"].ecranVideo.source=datas[i].source;
    contenu_mc["v"+i+"_mc"].rotationY=datas[i].rotationY;
    contenu_mc["v"+i+"_mc"].z=datas[i].z;
    contenu_mc["v"+i+"_mc"].ecranVideo.stop();
    contenu_mc["v"+i+"_mc"].ecranVideo.scaleMode="exactFit";
    contenu_mc["v"+i+"_mc"].ecranVideo.autoPlay=false;
}

//----- actions
// mur d'image 3D
contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
function murImages (evt:Event) {
    contenu_mc.rotationY=(mouseX-demieScene)*0.1;
    contenu_mc.rotationX=(mouseY-hauteurScene)*-0.1;
}

// interactivité vidéo
contenu_mc.addEventListener(MouseEvent.CLICK,zoomVideo);
function zoomVideo (evt:MouseEvent) {
    //
    contenu_mc.removeEventListener(Event.ENTER_FRAME,murImages);
    TweenMax.to(contenu_mc, 3, {rotationX:0, rotationY:0, delay:0,
    ➤ ease:Elastic.easeInOut});
    //
    for (var i:Number=0; i<contenu_mc.numChildren; i++) {
        Tween3D=TweenMax.to(contenu_mc["v"+i+"_mc"], 0.3, {z:datas[i].z,
    ➤ x:datas[i].x, y:datas[i].y, rotationY:datas[i].rotationY, delay:0,
    ➤ ease:Strong.easeInOut});
        TweenMax.to(contenu_mc["v"+i+"_mc"].ecranVideo, 2, {width:220,
    ➤ height:123, x:-110, y:-62, delay:0.3, ease:Elastic.easeOut});
    }
    //
    Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFini);
    function restaureFini (yo:TweenEvent) {
        contenu_mc.addChild(MovieClip(evt.target));
    }
    if (nomImageActive!=evt.target.name) {
        nomImageActive=evt.target.name
    }
}
```

```

//
TweenMax.to(evt.target.ecranVideo, 2, {width:440, height:246, x:-220,
➤ y:-123, delay:0.3, ease:Elastic.easeOut});
TweenMax.to(evt.target,2, {z:-100, x:0, y:0, rotationY:0, delay:0.3,
➤ ease:Elastic.easeOut});
} else {
    nomImageActive=" ";
    contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
}
}

// rollOvers
contenu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    evt.target.ecranVideo.play();
    evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    evt.target.ecranVideo.stop();
    evt.target.filters=[haloOut];
}

```

Dans la première partie du code, nous commençons par instancier l'ensemble des propriétés de chaque vidéo à travers un tableau, grâce à la méthode `Array`. Pour chaque nouvelle entrée, nous ouvrons et fermons une paire d'accolades. Chaque bloc créé enregistre, sous la forme de paramètres, chacune des propriétés et valeurs dont nous souhaitons disposer par la suite dans les fonctions d'animation :

```

var datas:Array = [
    {source:"video3D/3d-gKaster-C19.f4v",rotationY:-30,x:-245,y:-72,z:-25},
    {source:"video3D/3d-gKaster-amusement.f4v",rotationY:0,x:0,y:-72,z:40},
    {source:"video3D/3d-gKaster-balistic.f4v",rotationY:30,x:245,y:-72,z:-25},
    {source:"video3D/3d-gKaster-demoreel.f4v",rotationY:-30,x:-245,y:72,z:-25},
    {source:"video3D/3d-galaxieFull.f4v",rotationY:0,x:0,y:72,z:40},
    {source:"video3D/3d-particules.f4v",rotationY:30,x:245,y:72,z:-25}
];

```

Puis, nous utilisons une boucle `for` pour affecter, aux différents objets, les propriétés stockées dans le précédent tableau :

```

for (var i:Number=0; i<contenu_mc.numChildren; i++) {
    contenu_mc["v"+i+"_mc"].ecranVideo.source=datas[i].source;
    contenu_mc["v"+i+"_mc"].rotationY=datas[i].rotationY;
    contenu_mc["v"+i+"_mc"].z=datas[i].z;
    contenu_mc["v"+i+"_mc"].ecranVideo.stop();
    contenu_mc["v"+i+"_mc"].ecranVideo.scaleMode="exactFit";
    contenu_mc["v"+i+"_mc"].ecranVideo.autoPlay=false;
}

```

Dans ce contexte, pour cibler chaque objet individuellement, nous désignons d'abord le conteneur principal (`contenu_mc`), suivi de ses éléments descendants grâce à une syntaxe tabulée (avec des crochets []). En paramètre de cette structure, nous appelons les objets par leur nom d'occurrence en utilisant la valeur de `i` pour les distinguer les uns des autres

(contenu_mc["v"+i+"_mc"])). Il ne reste alors qu'à appliquer les valeurs relatives à chaque propriété initialement définie dans le tableau `datas`, pour affecter directement l'ensemble des objets. Ce qui donne, par exemple, pour le chemin de référence de chaque fichier vidéo désigné par la propriété `source` :

```
contenu_mc["v"+i+"_mc"].ecranVideo.source=datas[i].source;
```

Lorsque nous avons besoin de faire référence à chaque propriété de chacun des objets, nous invoquons également, dans les animations, le nom du tableau (`datas`) suivi du numéro d'ordre d'apparition de l'objet ciblé (avec `[i]`) et de la propriété concernée (par exemple `.z`). Ce qui donne pour la propriété `z` :

```
datas[i].z
```

Nous avons ensuite remplacé la répétition des interpolations `TweenMax` par une animation type, répétée dans une boucle `for`. Pour cibler chaque objet individuellement, nous reprenons le mécanisme abordé pour l'affectation des propriétés, dans la première partie du programme. Ce qui donne :

```
contenu_mc["v"+i+"_mc"]
```

À chaque itération des boucles `for`, ce sont tous les objets enregistrés qui sont affectés. Pour mettre à jour les données, il suffit donc de modifier les valeurs stockées dans le tableau initial (`datas`). L'ensemble de la construction est instantanément mise à jour.

À retenir

- Afin d'optimiser les ressources d'affichage, si des vidéos sont déployées, il est souhaitable de ne pas les jouer toutes simultanément et de les organiser pour un affichage frontal.
- Les composants vidéos ne disposent pas des mêmes propriétés que les `MovieClip`. Il est possible d'animer ces composants dans des `MovieClip` pour disposer de plus de contrôles sur chacun des objets.
- Pour optimiser le stockage de valeurs et simplifier la distribution des instructions sur plusieurs objets simultanément, nous utilisons deux types de structure : les tableaux et une boucle `for`. Les tableaux permettent le stockage des propriétés. La boucle simplifie leur affectation.
- Afin de cibler dynamiquement un symbole d'un conteneur parent, nous pouvons le capturer en désignant d'abord le conteneur parent, et à l'aide de crochets, nous y inscrivons en paramètre le nom de l'objet enfant.

Navigation spatiale 3D façon TimeMachine ou Aero

Puisque la 3D offre une profondeur `Z`, nous pouvons aussi distribuer les contenus de l'arrière-plan vers le premier plan comme le proposent, par exemple, les systèmes de navigation connus que sont `Aero` pour `Windows` et `TimeMachine` pour `Apple` (voir Figure 9.26).

Dans cette section, nous allons voir comment réaliser une navigation sur l'axe `Z`. Pour cela, nous utilisons différents `MovieClip` placés dans le scénario.

Figure 9.26

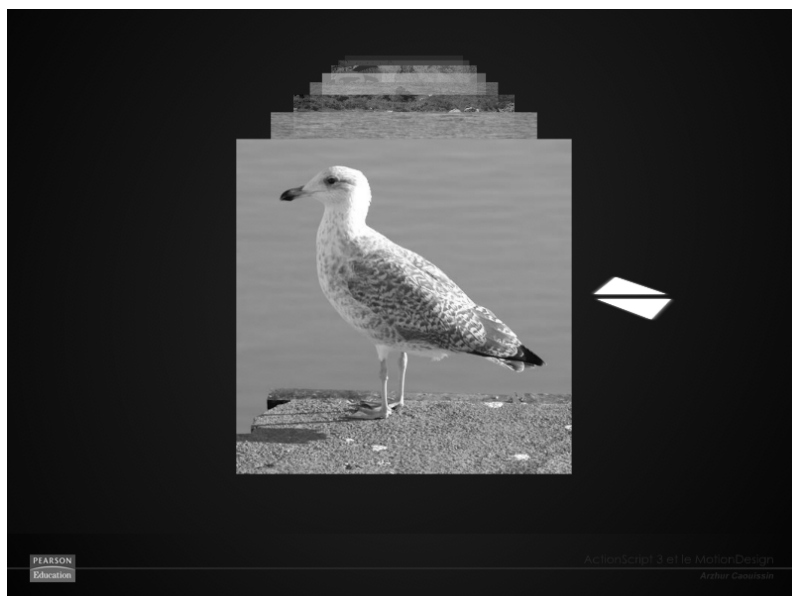
Aperçu du système d'archivage Apple Time Machine.



En cliquant sur une flèche, nous modifions leur position en Z avec une transition. Mais nous ajoutons au dispositif un système de boucle qui permet aussi de replacer les images déjà visitées, à l'arrière de la file. Nous ajoutons également un effet d'opacité selon la profondeur des objets. Nous déclinons enfin le principe en sens inverse avec une flèche qui oriente le mouvement dans le sens opposé (voir Figure 9.27).

Figure 9.27

Aperçu du document publié.



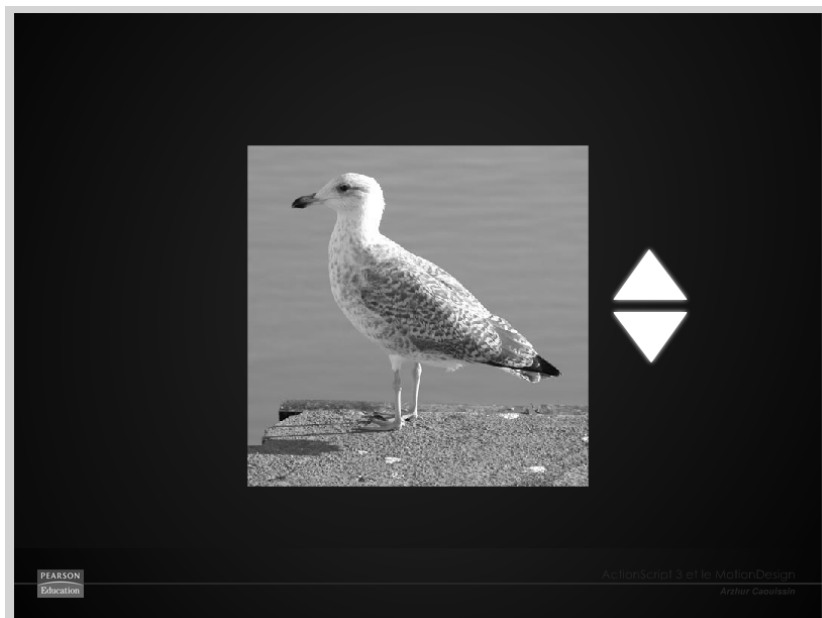


Exemples > ch9_3DNative_5.fla

Sur la scène principale du document "ch9_3DNative_5.fla", nous pouvons voir un symbole contenu_mc qui intègre une série de neuf MovieClip. Chacun dispose d'un nom d'occurrence et est réparti vers les calques, dans l'ordre d'affichage qui correspond à l'ordre d'empilement dans l'espace. Un symbole navigation_mc contient également deux flèches, haut et bas, pour faire avancer et reculer les images sur l'axe Z (voir Figure 9.28).

Figure 9.28

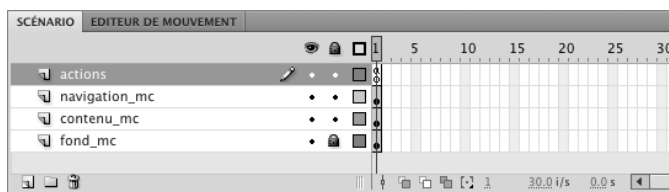
Aperçu de la scène principale.



Dans la fenêtre de scénario, nous identifions les symboles contenu_mc et navigation_mc (voir Figure 9.29).

Figure 9.29

Aperçu du scénario.



La fenêtre d'actions affiche le code suivant :

```
//----- initialisation
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;
```

```

var tween3DAvant:TweenMax;
var tween3DArriere:TweenMax;

contenu_mc.photo1_mc.z=1600;
contenu_mc.photo2_mc.z=1400;
contenu_mc.photo3_mc.z=1200;
contenu_mc.photo4_mc.z=1000;
contenu_mc.photo5_mc.z=800;
contenu_mc.photo6_mc.z=600;
contenu_mc.photo7_mc.z=400;
contenu_mc.photo8_mc.z=200;
contenu_mc.photo9_mc.z=0;

navigation_mc.rotationX=-90;

//----- actions

// point de fuite
var pointDeFuite:Point=new Point(stage.stageWidth/2,0);
transform.perspectiveProjection.projectionCenter=pointDeFuite;

// Avancer
navigation_mc.flecheHaut_btn.addEventListener(MouseEvent.CLICK,avancer);
function avancer(evt:MouseEvent) {
    for (var i:Number=0; i<contenu_mc.numChildren-1; i++) {
        TweenMax.to(contenu_mc.getChildAt(i), 0.5,
➤ {z:contenu_mc.getChildAt(i).z-200, alpha:1-((contenu_mc.getChildAt(i).z)/
➤ 1600), delay:0, ease:Strong.easeInOut});
    }
    if (contenu_mc.getChildAt(contenu_mc.numChildren-1).z<1) {
        tween3DAvant=TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-
➤ 1),0.5,{alpha:0,delay:0,ease:Strong.easeOut});
    }
    tween3DAvant.addEventListener(TweenEvent.COMPLETE,sortie);
}
//
function sortie(evt:TweenEvent) {
    contenu_mc.getChildAt(contenu_mc.numChildren-1).z=1600;
    contenu_mc.addChildAt(contenu_mc.getChildAt(contenu_mc.numChildren-1),0);
}

// Reculer
navigation_mc.flecheBas_btn.addEventListener(MouseEvent.CLICK,reculer);
function reculer(evt:MouseEvent) {
    for (var j:Number=0; j<contenu_mc.numChildren; j++) {
        tween3DArriere=TweenMax.to(contenu_mc.getChildAt(j), 0.5,
➤ {z:contenu_mc.getChildAt(j).z+200, alpha:1-((contenu_mc.getChildAt(j).z)
➤ /1600), delay:0, ease:Strong.easeInOut});
    }
    tween3DArriere.addEventListener(TweenEvent.COMPLETE,entree);
}

```

```
//
function entree (evt:TweenEvent) {
    contenu_mc.getChildAt(0).alpha=0;
    contenu_mc.addChildAt(contenu_mc.getChildAt(0),contenu_mc.numChildren-1);
    //
    var nombreDeBoucle:Number=1;
    var dureeBoucle:Number=100;
    var boucle:Timer=new Timer(dureeBoucle,nombreDeBoucle);
    boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
    boucle.start();
}
//
function lancerBoucle (evt:TimerEvent) {
    contenu_mc.getChildAt(contenu_mc.numChildren-1).z=0;
    TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-1), 0.5, {alpha:1,
    ➡ delay:0, ease:Strong.easeInOut});
}

// initialiser les alphas
for (var k:Number=0; k<contenu_mc.numChildren-1; k++) {
    contenu_mc.getChildAt(k).alpha=1-((contenu_mc.getChildAt(k).z)/1600);
}

```

Le programme est structuré en cinq parties dont : l'initialisation, la gestion du point de fuite, la navigation avec la flèche du haut, puis avec la flèche du bas, et enfin, la gestion de l'opacité au lancement de l'application. L'ordre d'exécution et les valeurs de calcul n'étant pas les mêmes pour les deux flèches, nous avons préféré isoler ici les deux fonctions.

D'abord, nous définissons les classes et quelques variables requises pour les transitions. Nous modifions ensuite l'agencement des contenus dans l'espace.

Plus loin, dans les actions, nous redéfinissons la position du point de fuite, par défaut placé au centre de la scène. Dans notre configuration, nous voulons surplomber légèrement l'animation de manière à percevoir les objets, de même dimensions, situés à l'arrière-plan des premiers éléments. Pour ce faire, nous utilisons la propriété `filedOfView` de la classe `perspectiveProjection` :

```
// point de fuite
var pointDeFuite:Point=new Point(stage.stageWidth/2,0);
transform.perspectiveProjection.projectionCenter=pointDeFuite;

```

Le point de fuite est désigné par un objet `Point` qui possède deux valeurs, `x` et `y` (vu lors de la programmation de squelettes).

Le principe du point de fuite, en perspective, est de situer le point de départ de tous les tracés fuyants qui partent de la ligne d'horizon. Plus le point de fuite est haut, plus vous dominez la scène. Plus vous prenez, en somme, de l'altitude.

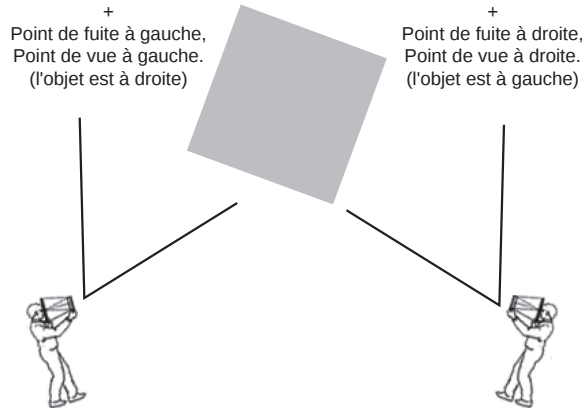
Vous pouvez aussi placer le point de fuite horizontalement. Cela détermine la direction de la perspective. Si le point de fuite est situé à gauche (de valeur `x` faible), vous vous placez alors à gauche de la scène et l'objet se trouve sur votre droite. Si le point de fuite se trouve à droite, votre regard aussi se retrouve à droite et l'objet, lui, se retrouve à gauche (voir

Figure 9.30). Vous placez le point de fuite en fonction du côté de l'objet que vous souhaitez observer.

Le point de fuite représente aussi le reflet de la position de votre œil sur la ligne d'horizon, autrement dit, du point de vue.

Figure 9.30

Placement du point de fuite.



Propriétés 3D de la scène

La scène comporte plusieurs propriétés qui peuvent être contrôlées en ActionScript, y compris en animation. Ces propriétés sont : le champ de vision, le point de fuite et la focale.

- Le champ de vision se définit entre 0 et 180 degrés. Le code suivant applique un champ de vision de 55°, qui est aussi la valeur appliquée par défaut.

```
var champDeVision:Number=55;
this.transform.perspectiveProjection.fieldOfView=champDeVision;
```

- Le point de fuite désigne le point de départ de tout tracé de forme perspective, à partir de la ligne d'horizon. Il représente la position de l'œil du spectateur. Le code suivant indique la valeur appliquée par défaut dans un document Flash 3D :

```
var pointDeFuite:Point = new Point( stage.stageWidth/2, stage.stageHeight/2);
transform.perspectiveProjection.projectionCenter = pointDeFuite;
```

- La focale indique le type de déformation appliqué aux fuyantes, déterminé par la distance entre l'écran et l'objet. La focale est calculée dynamiquement, comme suit :

```
var focale:uint=stage.stageWidth/ 2 * ( Math.cos(champDeVision/2) /
    ➤ Math.sin(champDeVision/2) );
transform.perspectiveProjection.focalLength= focale;
```

- Vous pouvez, par exemple, modifier la focale ou toute autre propriété de la scène 3D, en la manipulant à travers une instruction, comme celle-ci où les valeurs utilisées reprennent la position courante du pointeur en X :

```
addEventListener(Event.ENTER_FRAME,focaleTest);
function focaleTest(evt:Event) {
    transform.perspectiveProjection.focalLength=mouseX;
}
```

Le défilement des images de l'arrière vers le premier plan est activé par la flèche du haut. Les actions associées à ce comportement sont rassemblées dans la partie "Avancer" :

```
// Avancer
navigation_mc.flecheHaut_btn.addEventListener(MouseEvent.CLICK,avancer);
function avancer(evt:MouseEvent) {
    for (var i:Number=0; i<contenu_mc.numChildren-1; i++) {
        TweenMax.to(contenu_mc.getChildAt(i), 0.5,
        {z:contenu_mc.getChildAt(i).z-200, alpha:1.1-((contenu_mc.getChildAt(i).z)
➤ /1600), delay:0, ease:Strong.easeInOut});
    }
    // autres actions
}
```

Dans un premier temps, la fonction avancer est exécutée sur un clic de souris. Dans cette fonction, nous mettons en place une boucle for qui applique une interpolation TweenMax pour l'ensemble des éléments contenus dans le clip principal (avec contenu_mc.getChildAt(i)), comme vu précédemment. La transition réduit ici l'index z d'une valeur de 200 pixels, pour chaque objet de la liste d'affichage et les fait ainsi progresser, simultanément, vers l'écran.

Dans le même temps, un alpha permet de modifier aussi l'opacité des objets en fonction de leur index respectif :

```
alpha:1.1-((contenu_mc.getChildAt(i).z)/1600),
```

La valeur 1 600 correspond au seuil que nous avons arbitrairement fixé, pour le démarrage du défilement, en plaçant le premier objet à cette profondeur (voir les paramètres d'initialisation en début de programme).

Dans cette équation, plus l'index est proche de 1 600, plus l'alpha diminue. Inversement, plus il tend vers 0, plus l'objet se révèle.

Pour comprendre plus précisément le calcul, nous savons que l'alpha se mesure sur une échelle de 0 à 1. Or, l'index maximum que nous observons est 1 600. Il n'est pas possible d'appliquer un alpha de 1 600 en attribuant systématiquement la valeur de l'index z comme valeur d'alpha. L'équation reprend donc l'index z de chaque objet et le divise par la profondeur z maximum pour obtenir une valeur comprise entre 0 et 1.

Mais, ce faisant, nous obtenons l'inverse de ce que nous désirons. L'alpha est nul pour les images de premier plan et vaut presque 1 pour les images situées à l'arrière. Pour inverser les valeurs, nous effectuons donc une petite soustraction à partir de la valeur d'alpha maximum : 1. Enfin, pour garantir toutefois que les premières images situées en premier plan demeurent intégralement visibles et ne risquent pas d'être légèrement translucides (d'alpha légèrement inférieur à 1), nous augmentons la valeur à 1.1.

À la suite, dans la même fonction, nous plaçons quelques structures conditionnelles pour déterminer quels objets de la liste doivent être remplacés au premier-plan :

```
function avancer(evt:MouseEvent) {
    for (var i:Number=0; i<contenu_mc.numChildren-1; i++) {
        TweenMax.to(contenu_mc.getChildAt(i), 0.5,
➤ {z:contenu_mc.getChildAt(i).z-200, alpha:1.1-((contenu_mc.getChildAt(i).z)/
➤ 1600), delay:0, ease:Strong.easeInOut});
    }
}
```

```

    }
    if (contenu_mc.getChildAt(contenu_mc.numChildren-1).z<1) {
        tween3DAvant=TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-
        ➤ 1),0.5,{alpha:0,delay:0,ease:Strong.easeOut});
    }
    tween3DAvant.addEventListener(TweenEvent.COMPLETE,sortie);
}
//
function sortie(evt:TweenEvent) {
    contenu_mc.getChildAt(contenu_mc.numChildren-1).z=1600;
    contenu_mc.addChildAt(contenu_mc.getChildAt(contenu_mc.numChildren-1),0);
}

```

Une condition vérifie d'abord les objets dont l'index est inférieur à 1, c'est-à-dire que nous filtrons uniquement les objets situés au premier plan. Pour ces objets, nous ajoutons une interpolation qui le fait disparaître avec l'alpha passé à 0.

Dès que la transition (tween3DAvant) est terminée (COMPLETE), un écouteur exécute une autre fonction (sortie). Nous modifions alors l'index z de l'objet situé au sommet de la liste d'affichage, c'est-à-dire, en premier plan, et le ramenons à 1 600, donc, à une échelle qui le représente à l'arrière-plan. Mais, comme nous le savons, la 3D dans Flash est limitée à chaque objet. Il faut donc, en plus, modifier l'ordre d'empilement de l'objet dans la liste d'affichage. La deuxième instruction prend donc l'objet situé au sommet de la pile (contenu_mc.getChildAt(contenu_mc.numChildren-1)) pour le replacer au-dessous (0).

Les actions exécutées pour la flèche du bas sont une déclinaison de la flèche du haut, à ceci près que les valeurs z et alpha sont inversées.

Nous relevons néanmoins que, par nécessité, nous utilisons un chronomètre (Timer) pour décaler, dans le temps, les deux méthodes qui interviennent sur l'ordre d'affichage (addChild() et getChild()) afin d'éviter les conflits. En exécutant les deux instructions simultanément, selon le contexte et le poids des contenus affichés dans les objets, le repositionnement à l'index z pourrait effectivement ne pas avoir lieu, écrasé par la méthode addChild(), prioritaire et peut-être toujours en cours d'exécution :

```

//
function entree (evt:TweenEvent) {
    contenu_mc.getChildAt(0).alpha=0;
    contenu_mc.addChildAt(contenu_mc.getChildAt(0),contenu_mc.numChildren-1);
    //
    var nombreDeBoucle:Number=1;
    var dureeBoucle:Number=100;
    var boucle:Timer=new Timer(dureeBoucle,nombreDeBoucle);
    boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
    boucle.start();
}
//
function lancerBoucle (evt:TimerEvent) {
    contenu_mc.getChildAt(contenu_mc.numChildren-1).z=0;
    TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-1), 0.5, {alpha:1,
    ➤ delay:0, ease:Strong.easeInOut});
}

```

Le programme s'achève avec l'initialisation de l'alpha au démarrage de l'application. Ceci afin d'appliquer, dès le début, une transparence aux objets situés en profondeur :

```
// initialiser les alphas
for (var k:Number=0; k<contenu_mc.numChildren-1; k++) {
    contenu_mc.getChildAt(k).alpha=1-((contenu_mc.getChildAt(k).z)/1600);
}
```

Dans le fichier SWF, les deux flèches actionnent le défilement, dans un sens comme dans l'autre, en préservant toujours l'alpha et les index de chaque objet.

**Info**

Pour en savoir plus sur la 3D dans Flash, et notamment sur la gestion de formes matricielles (volumes composés de triangles), consultez l'adresse suivante : http://help.adobe.com/fr_FR/ActionScript/3.0/ProgrammingAS3/WSF24A5A75-38D6-4a44-BDC6-927A2B123E90.html. D'autres techniques permettent aussi de gérer la 3D à partir de formes déjà modelées. Nous les abordons au chapitre suivant.

À retenir

- Vous pouvez contrôler la focale et le point de fuite d'une perspective 3D grâce à la propriété `perspectiveProjection`.
- L'ordre d'affichage des objets détermine la cohérence d'un système de navigation spatial. Nous utilisons la méthode `addChild()` afin de redistribuer les objets selon leur position sur l'index z.

Synthèse

Dans ce chapitre, vous avez appris à créer des présentations 3D à partir des classes natives de Flash, compatibles avec Flash 10 (CS4) et versions ultérieures. La 3D dans Flash est simple, mais, ne permet pas encore d'importer directement des objets modélisés en 3D sans recourir à des classes externes. Elle apparaît malgré tout en voie de standardisation. Le lecteur 10 est, fin 2009, déjà implanté en Europe à près de 92 % des postes utilisateurs disposant d'un lecteur Flash. La 3D reste donc à ce jour largement compatible avec les configurations existantes des utilisateurs (source : http://www.adobe.com/products/player_census/flashplayer/version_penetration.html). Seule la solidité de la configuration (ressources graphiques et bandes passantes) peuvent en tempérer encore la diffusion.

10

La 3D et *PaperVision*

Introduction

Si nous pouvons créer des interfaces 3D à partir d'objets 2D, dans Flash, il n'est pas encore possible d'y importer des objets 3D réels et d'interagir nativement avec eux. Cependant, des classes ActionScript libres sont disponibles pour la gestion de la 3D réelle au sein de l'environnement Flash. Certaines – comme *PaperVision* – sont compatibles avec d'anciennes versions de Flash, pour AS3 et AS2 et Flash 9 (et les versions suivantes).

Dans ce chapitre, nous allons voir comment installer *PaperVision* pour AS3 sous Windows et Mac OS X, ainsi que la manière d'interagir sur des animations d'objets 3D réels exportés depuis l'application libre Google Sketchup ou tout autre logiciel 3D. Nous allons également découvrir comment contrôler des environnements 3D, créés à partir de simples primitives avec le clavier pour simuler la navigation dans un espace en trois dimensions.

À l'issue de ce chapitre, vous serez en mesure de créer des visionneuses d'objets 3D et des galeries 3D.

Installer *PaperVision*

La classe *PaperVision* rassemble une série de fonctionnalités 3D, regroupées dans un même répertoire. Pour en disposer, nous devons d'abord télécharger cette classe. Une fois chargée sur votre ordinateur, elle doit être copiée dans le répertoire de votre site, au même niveau que les documents Flash qui constituent votre projet. Si, pour automatiser le processus, nous souhaitons centraliser la gestion des classes, nous devons redéfinir les préférences de Flash. Nous détaillons cette procédure plus loin dans ce chapitre.

Une fois dans Flash, pour invoquer les sous-classes *PaperVision*, qu'elles aient été placées localement ou intégrées à l'API, nous y faisons référence directement depuis la fenêtre des Actions. À la compilation, Flash intègre les scripts utilisés, requis pour la gestion des commandes 3D. Une fois déployées, elles n'ont donc pas besoin d'être placées sur le serveur. Seuls les objets 3D et les textures éventuellement appelées par le code doivent accompagner le document SWF.

Dans ce chapitre, nous développons les commandes 3D de *PaperVision* directement dans le scénario. De nombreux développeurs adoptent la méthode externalisée sous la forme de *packages* (classes .as). Nous verrons, dans ce chapitre, qu'il n'est pas nécessaire de recourir à une externalisation du code pour exécuter *PaperVision*.

La classe *PaperVision* étant constituée de nombreux fichiers et de sous-dossiers, vous trouverez plus facile d'accéder à une version zippée de cette classe. Vous en trouverez une sur le site Googlecode à l'adresse : <http://code.google.com/p/papervision3d/downloads/list>.

Vous pouvez aussi la télécharger en suivant directement le lien qui correspond à la dernière version publiée au moment où nous éditons cet ouvrage : http://papervision3d.googlecode.com/files/Papervision3D_2.1.920.zip.

Bien que cette méthode de chargement soit simple, de nombreux utilisateurs préfèrent gérer le téléchargement des classes à l'aide d'utilitaires capables d'en assurer la mise à jour, lorsque la source distante a été modifiée par exemple.

Nous vous proposons, pour répondre à de nombreuses questions en rapport avec cette technique de chargement, de présenter ces utilitaires dans les deux sections suivantes (Tortoise SVN pour Windows et SVNX pour Mac OS X). Pour les lecteurs qui auront téléchargé PaperVision à partir du format zippé, vous pouvez passer directement à la section "Intégrer la classe PaperVision à Flash".

Téléchargement avec Tortoise SVN pour Windows

L'utilitaire de chargement de dossiers nommé Tortoise SVN, compatible Windows, est disponible gratuitement à l'adresse suivante : <http://tortoisesvn.tigris.org/>.

Pour installer cet utilitaire, procédez comme suit :

1. Sur le site <http://tortoisesvn.tigris.org/>, cliquez directement sur l'onglet Downloads. En bas de la page de chargement, apparaît un tableau affichant toutes les versions disponibles, filtrées par langue et version de système (32 bits ou 64 bits)

Figure 10.1

Téléchargement de l'utilitaire Tortoise SVN pour Windows.

Ads by Google

Language packs

	Country	32 Bit	64 Bit	Separate manual (PDF)	
1	Bulgarian	Setup	Setup		
2	Catalan	Setup	Setup		
3	Chinese, simplified	Setup	Setup	TSVN	TMerge
4	Chinese, traditional	Setup	Setup		
5	Croatian	Setup	Setup		TMerge
6	Czech	Setup	Setup		
7	Danish	Setup	Setup		
8	Dutch	Setup	Setup		TMerge
9	Finnish	Setup	Setup	TSVN	TMerge
10	French	Setup	Setup	TSVN	TMerge
11	Georgian	Setup	Setup		
12	German	Setup	Setup	TSVN	TMerge

2. Cliquez sur le lien Setup correspondant à votre version de système. Le chargement s'effectue. Des supports au format PDF sont également disponibles en français : Separate manual (PDF). Vous pouvez, si besoin, les télécharger.
3. Installez le logiciel en double-cliquant sur l'icône d'installation chargée sur votre poste de travail. Puis validez toutes les étapes.

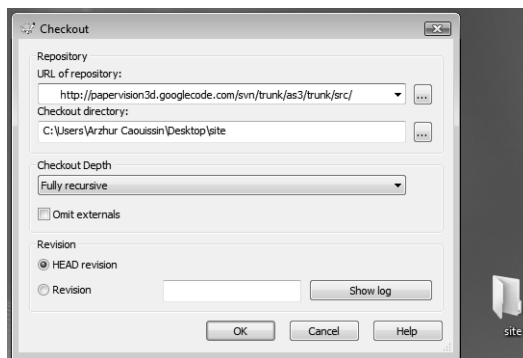
Une fois installé, vous le retrouvez dans le dossier Program files et dans le menu Démarrer > Applications. Le principe de l'utilitaire Tortoise SVN est de rendre ses fonctions accessibles par un simple clic-droit sur le répertoire de votre choix. Vous pouvez alors y importer un dossier distant à partir d'une URL, mettre à jour le contenu éventuellement déjà chargé :

4. Directement, sur le dossier racine de votre projet, faites clic-droit > SVN CheckOut.

5. Dans la boîte de dialogue, saisissez l'URL de la classe PaperVision – ou d'une autre classe à importer : <http://papervision3d.googlecode.com/svn/trunk/as3/trunk/src/> (voir Figure 10.2).
6. Puis cliquez sur OK.

Figure 10.2

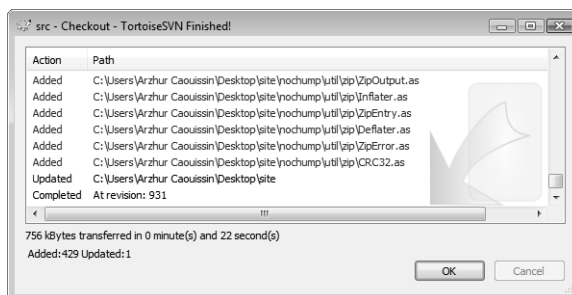
Importer la classe avec TortoiseSVN sous Windows.



7. Une fenêtre de chargement s'affiche. Patientez. Une fois le chargement terminé, cliquez à nouveau sur OK (voir Figure 10.3).

Figure 10.3

Chargement de la classe PaperVision.



8. À l'intérieur de votre dossier, la classe PaperVision a été importée. Elle apparaît sous la forme de deux répertoires : "nochump" et "org" (voir Figure 10.4).

Figure 10.4

Classe chargée.



La classe est disponible dans votre projet. Vous pouvez l'invoquer directement depuis votre document Flash.

Téléchargement avec SVNX pour Mac OS X

L'utilitaire de chargement de dossiers, nommé SVNX et compatible Mac OS X, requiert deux installations : le plug-in SCPlugin et l'utilitaire SVNX.

Le plug-in SCPlugin est disponible à cette adresse : <http://scplugin.tigris.org/servlets/ProjectDocumentList>. L'utilitaire SVNX ici : <http://www.lachoseinteractive.net/en/community/subversion/svnx/download/>.

1. Pour installer d'abord le plug-in, saisissez dans votre navigateur l'adresse : <http://scplugin.tigris.org/servlets/ProjectDocumentList>.
2. Puis, cliquez sur la dernière version de l'installeur proposée dans la liste de téléchargement (voir Figure 10.5).

Figure 10.5

Téléchargement du plug-in SCPlugin pour Mac OS X.

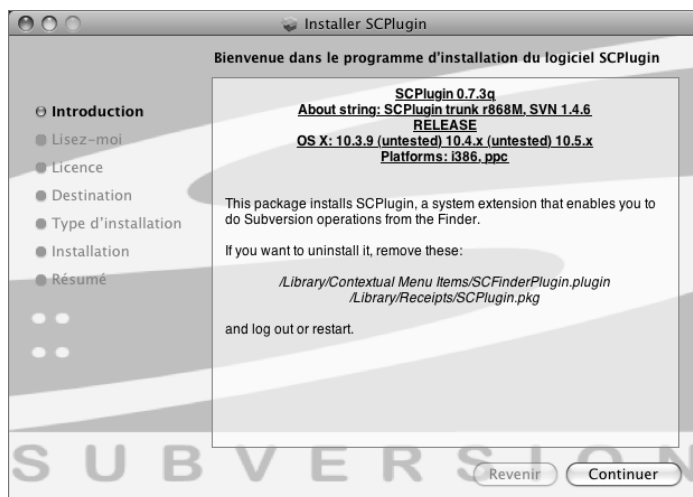


Name	Status	Modified by	Size	Reservations	Description
SCPlugin-0.7.3q-SVN.1.4.6.dmg	Stable	jackreppening on Thursday, May 21, 2009 at 7:22:49 PM	2.53 MB		SCPlugin-0.7.3q-SVN.1.4.6.dmg
SCPlugin-0.7.3q-SVN.1.4.6.dmg.sig	Stable	jackreppening on Thursday, May 21, 2009 at 7:23:09 PM	233 bytes		SCPlugin-0.7.3q-SVN.1.4.6.dmg.sig
SCPlugin-0.7.3q-SVN.1.5.6.dmg	Stable	jackreppening on Thursday, May 21, 2009 at 6:59:14 PM	2.77 MB		SCPlugin-0.7.3q-SVN.1.5.6.dmg
SCPlugin-0.7.3q-SVN.1.5.6.dmg.sig	Stable	jackreppening on Thursday, May 21, 2009 at 6:58:43 PM	233 bytes		SCPlugin-0.7.3q-SVN.1.5.6.dmg.sig
SCPlugin-0.7.3q-SVN.1.6.2-1386.dmg	Stable	jackreppening on Tuesday, September 1, 2009 at 12:33:03 PM	3.44 MB		SCPlugin-0.7.3q-SVN.1.6.2-1386.dmg
SCPlugin-0.7.3q-SVN.1.6.2-1386.dmg.sig	Stable	jackreppening on Tuesday, September 1, 2009 at 12:33:27 PM	233 bytes		SCPlugin-0.7.3q-SVN.1.6.2-1386.dmg.sig

3. Le *package* téléchargé ouvre une fenêtre d'installation (voir Figure 10.6).

Figure 10.6

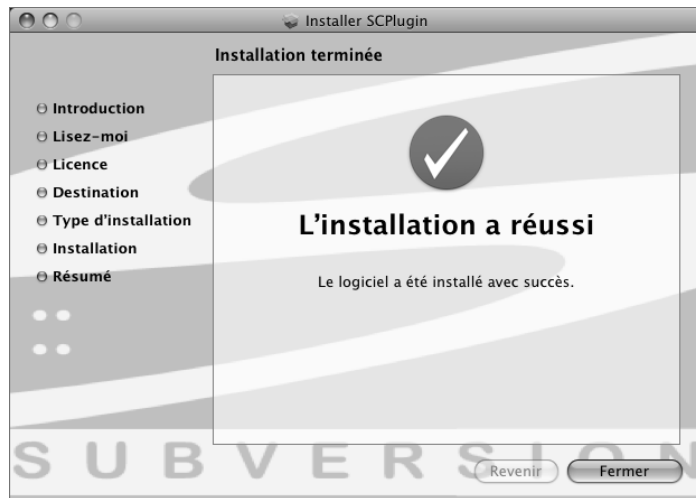
Installation du plug-in SCPlugin pour Mac OS X.



5. Validez toutes les étapes jusqu'à l'écran final (voir Figure 10.7).

Figure 10.7

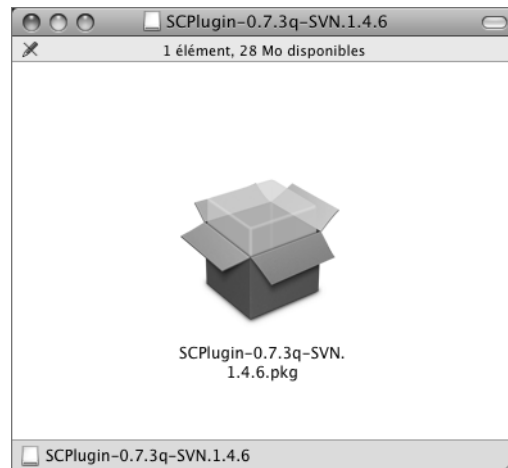
Confirmation de l'installation.



Le dossier du plug-in apparaît sur votre bureau (voir Figure 10.8).

Figure 10.8

Plug-in installé.



Vous pouvez maintenant installer l'utilitaire SVNX disponible à l'adresse suivante : <http://www.lachoseinteractive.net/en/community/subversion/svnx/download/>.

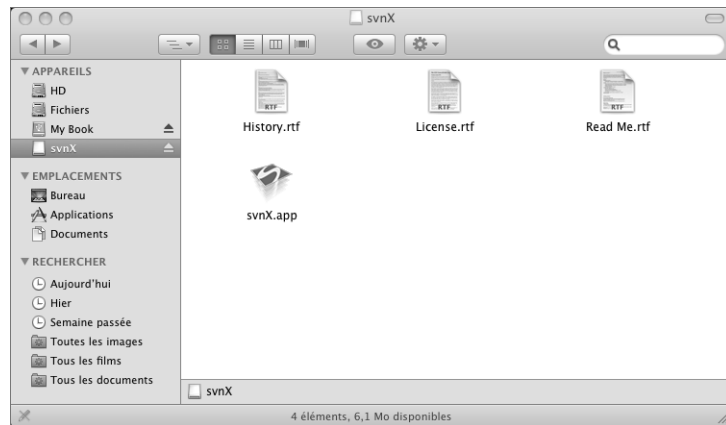
6. Dans la page de chargement, cliquez directement sur l'icône située en bas de page pour le chargement de l'application (voir Figure 10.9).
7. Une fois le chargement effectué, glissez-déposez directement l'application du répertoire, ouvert automatiquement, vers le dossier Applications de votre système. Pour organiser les fichiers, déposez-les dans un nouveau répertoire que vous nommerez SVNX (voir Figure 10.10).

Figure 10.9

Téléchargement de l'utilitaire SVNX pour Mac OS X.

**Figure 10.10**

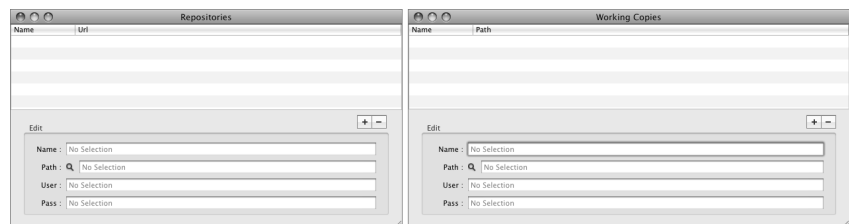
Téléchargement de l'utilitaire SVNX pour Mac OS X.



Une fois copié dans Applications, l'utilitaire est fonctionnel. Pour charger la classe Paper-Vision, *via* l'utilitaire SVNX : lancez l'utilitaire SVNX en double-cliquant dessus. Deux fenêtres apparaissent (voir Figure 10.11).

Figure 10.11

Ouverture de l'utilitaire SVNX.



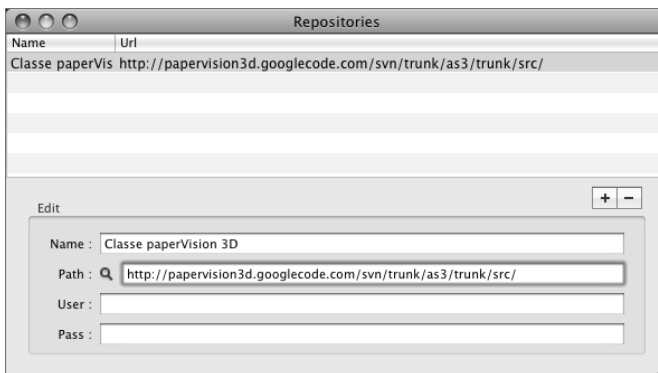
Seule la première fenêtre nous intéresse. Elle désigne l'URL à télécharger :

1. Pour définir un nouveau chargement, dans la fenêtre gauche, cliquez sur le bouton Plus.
2. Puis, renseignez un Nom (champ Name). Saisissez par exemple PaperVision3D.

3. Dans le champ Path, inscrivez l'adresse de la classe PaperVision à importer (voir Figure 10.12) : <http://papervision3d.googlecode.com/svn/trunk/as3/trunk/src/>.

Figure 10.12

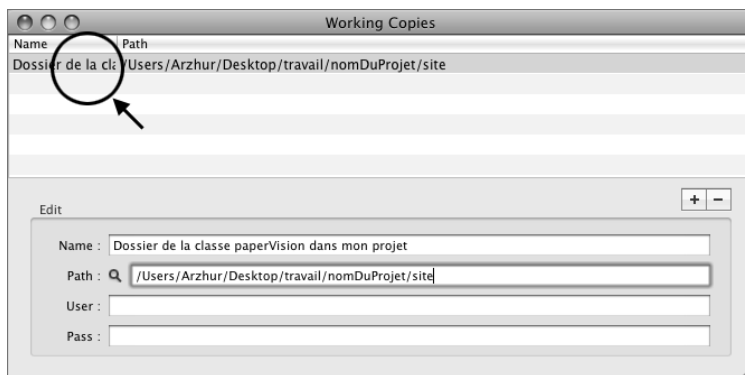
Définition du contenu à importer (fenêtre de gauche).



Le chargement du contenu appelé par l'URL se fait en double-cliquant sur la référence du chargement que nous venons de créer, dans la liste située au sommet de la première fenêtre (voir Figure 10.13).

Figure 10.13

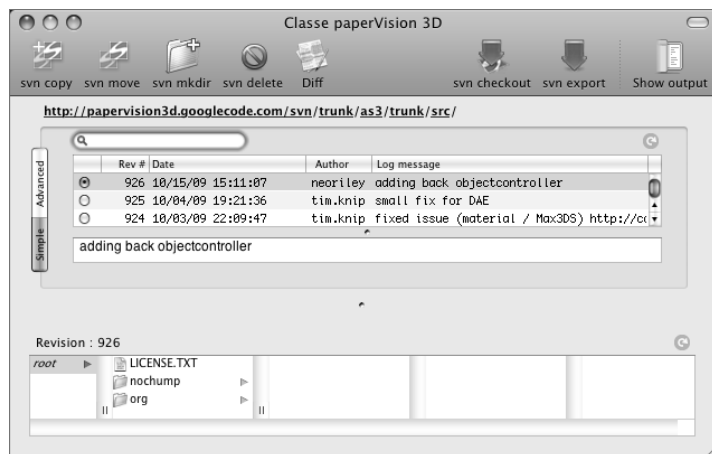
Ouverture de la fenêtre de chargement.



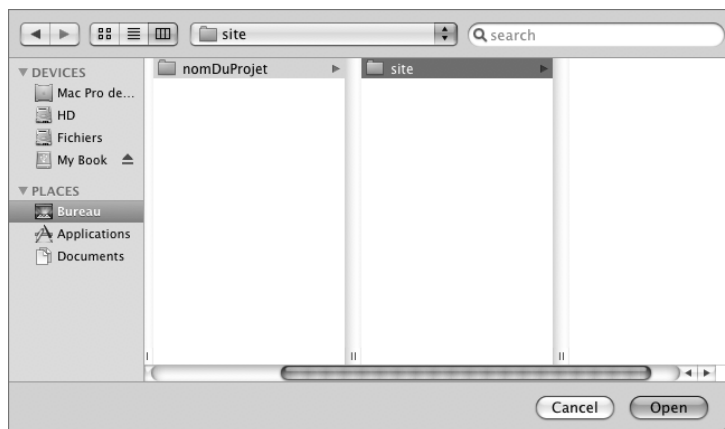
1. Double-cliquez sur la référence pour lancer le chargement. Une nouvelle fenêtre apparaît (voir Figure 10.14).
2. Pour définir un dossier cible et y importer la classe, cliquez sur le lien "svn checkout". Une fenêtre de sélection de dossier apparaît.
3. Sélectionnez le répertoire de destination, soit le dossier racine du site pour lequel vous importez la classe ou un dossier distinct utilisé pour une gestion globale et native des classes, comme nous le détaillons plus loin dans ce chapitre (voir Figure 10.15).
4. Validez pour activer directement le chargement des fichiers. Le chargement s'effectue jusqu'à ce que la barre de chargement située en bas de la fenêtre s'arrête (voir Figure 10.16).

Figure 10.14

Activation du chargement.

**Figure 10.15**

Fenêtre de sélection du dossier cible.

**Figure 10.16**

Chargement en cours.



5. Lorsque le chargement est terminé, quittez l'application. À l'intérieur de votre dossier, la classe PaperVision a été importée. Elle apparaît sous la forme de deux répertoires : nochump et org (voir Figure 10.17).

Vous pouvez maintenant l'appeler directement depuis un document Flash ou l'intégrer à l'API de Flash.

Figure 10.17

Classe chargée.



Intégrer la classe *PaperVision* à Flash

Vous pouvez placer la classe *PaperVision*, dans Flash, selon deux méthodes. Soit, vous intégrez le chemin de la classe dans le moteur de Flash pour la rendre accessible à partir de tout nouveau document. Soit vous la copiez ponctuellement et manuellement dans chaque nouveau projet.

Intégration native

Lorsque vous intégrez une classe nativement dans Flash, vous ne devez plus, par la suite, modifier l'emplacement du répertoire appelé en référence. Aussi, nous vous conseillons de bien choisir son emplacement avant de procéder à la liaison :

1. Par exemple, dans le dossier Flash du répertoire Applications de votre système (ou dans Program files pour Windows), créez un nouveau répertoire et nommez-le "Classes-Persos".
2. Dans ce nouveau dossier, créez un répertoire et intitulez-le à présent "papervision".
3. Copiez-y intégralement les dossiers que nous venons de télécharger ("org" et "nochump"). Attention toutefois, pour ne pas perdre la connexion avec Tortoise, préférez cibler directement ce nouveau répertoire lors du téléchargement de la classe.
4. Revenez dans Flash.
5. Pour intégrer la classe nativement, dans Flash, affichez les préférences (Cmd+U sur Mac ou Ctrl+U sur Windows).
6. Dans la catégorie ActionScript, cliquez sur le bouton Paramètres d'ActionScript 3.

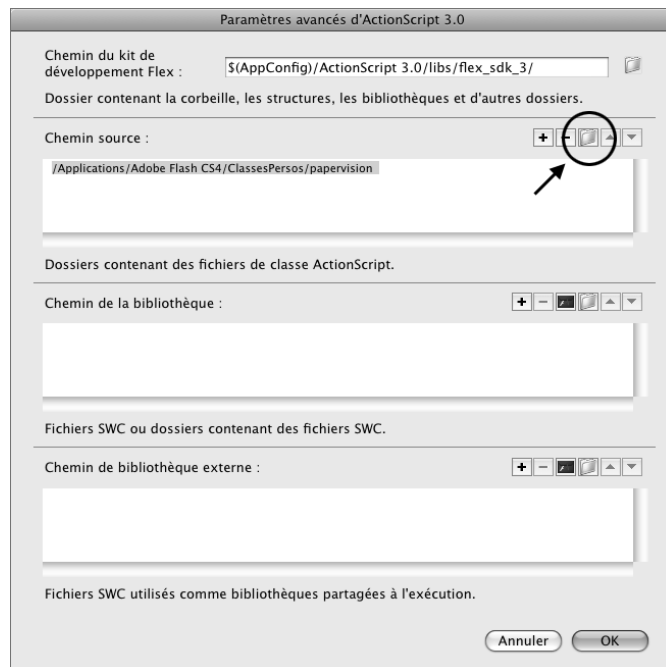
Dans la boîte de dialogue, trois zones de définition de chemin apparaissent. La première zone concerne l'intégration de classes natives (les trois zones sont confondues dans les anciennes versions de logiciel).

Pour la classe `PaperVision`, le dossier intitulé "org" contient l'ensemble des éléments requis. Plus précisément, c'est le dossier "papervision3d", contenu dans le dossier "org", contenu à son tour dans "papervision", qui devra plus tard être invoqué par ActionScript.

1. Dans la fenêtre de dialogue de Flash, à droite, cliquez sur le bouton de sélection de dossier. Puis, sélectionnez le répertoire "papervision" que nous venons de créer. En validant, le chemin de liaison apparaît dans la liste des classes intégrées (voir Figure 10.18).

Figure 10.18

Fenêtre d'intégration des classes natives dans Flash CS4.



2. Validez en refermant toutes les fenêtres. La classe est désormais intégrée nativement dans l'application et disponible pour tous les nouveaux documents.
3. Relancez de préférence l'application pour garantir la prise en compte de ces nouveaux éléments.

Intégration ponctuelle

Vous pouvez aussi placer ponctuellement la classe dans chaque projet pour lequel elle est requise. Il n'est donc, dans ce cas, pas nécessaire de l'intégrer nativement comme vu à la section précédente :

1. Copiez le répertoire "org" que nous venons de télécharger.
2. Collez ce répertoire directement à la racine de votre site.

La classe est prête à être utilisée pour les documents Flash placés uniquement à la racine de ce dossier. Vous pouvez commencer à réaliser des interfaces avec des objets 3D.

À retenir

- PaperVision est une classe externalisée qu'il faut télécharger avant de pouvoir l'exploiter. Des utilitaires permettent de charger cette classe dynamiquement : Tortoise SVN pour Windows et SVN X pour Mac OS X.
- Il est possible d'intégrer une liaison native de Flash avec avec PaperVision, ou toute autre classe en modifiant les paramètres d'ActionScript 3, depuis les préférences du logiciel. Il faut pour cela cibler le répertoire qui les contient depuis la fenêtre de liaison de classes.
- Si vous n'intégrez pas les classes nativement, vous devez les copier, à chaque utilisation, au même niveau d'arborescence que le document qui y fait référence.
- Les classes sont compilées dans le SWF à la publication. Il n'est pas nécessaire de les placer sur le serveur d'hébergement.

Modéliser en 3D pour Flash et *PaperVision*

Avant d'importer dans Flash des objets 3D réels, grâce à la classe PaperVision, vous devez naturellement disposer de ces objets. Vous pouvez les réaliser à partir de logiciels professionnels ou gratuits. Nous présentons ici différentes solutions pour rendre les créations 3D compatibles avec Flash et PaperVision. Nous proposons notamment l'outil Google Sketchup, logiciel 3D libre, compatible avec Flash et PaperVision.

Exporter la 3D avec les logiciels standard

Tous les logiciels 3D sont compatibles Flash et PaperVision dès lors qu'ils peuvent exporter au format DAE (Maya, 3DS Max, Cinema 4D, Blender, Swift,...). Pour de nombreux logiciels toutefois, le plug-in *Collada* est requis. Il est disponible sur Internet. Pour chaque logiciel, reportez-vous au lien en rapport :

- Blender requiert un plug-in. L'aide à l'installation de Collada pour Blender est disponible à cette adresse : <http://colladablender.illusoft.com/cms/content/blogcategory/25/29/>.
- Swift 3D gère nativement l'export au format DAE. Consultez le site suivant pour en savoir plus sur Swift 3D : <http://www.erain.com/products/swift3d/?erain=v6pr>.
- Cinema 4D R11 accepte également le format DAE de Collada : <http://www.maxon.net/products/cinema-4d.html>.
- 3DSMax et Maya disposent d'un plug-in Collada spécifique. Retrouvez-le à : <http://sourceforge.net/projects/colladamaya/>.
- Une aide plus générale sur l'exportation de fichiers 3D avec Collada, pour tous logiciels 3D, est disponible à l'adresse : <http://www.australopitech.com/883-collada-papervision3d>.



Limite de taille des fichiers 3D. La gestion de la 3D dans Flash avec PaperVision suppose que les fichiers puissent être consultés par tout type d'utilisateur et avec un chargement le plus rapide possible. Il est recommandé d'éviter de dépasser les scènes 3D composées de plus de 1 500 à 3 000 polygones. Au-delà, veillez à bien organiser le chargement du contenu par étapes successives pour éviter un chargement trop long, et éviter que le moteur de rendu ne peine à calculer entièrement le volume importé (présence de trous, points qui décrochent, etc.).

Exporter la 3D avec Google Sketchup

Google Sketchup est un logiciel de modélisation 3D, gratuit, si l'on s'en tient à une version basique. Cet outil initialement développé pour promouvoir la technologie GoogleEarth, offre une solution accessible et simple à utiliser pour créer des objets 3D pour l'environnement de Flash avec PaperVision, d'autant que Sketchup est très bien documenté – et en français. Google propose aussi une banque d'objets 3D déjà modélisés et libres de droits, simples à charger, pour un usage toutefois non commercial.



Les conditions d'utilisation de la banque d'objets 3D Google Sketchup sont disponibles à cette adresse : <http://sketchup.google.com/intl/fr/3dwh/tos.html>.

Pour être importés dans Flash, les objets 3D de Google doivent être exportés au format DAE. Mais, Sketchup ne permet d'exporter qu'aux formats KMZ (optimisé) ou SKP (format natif).

Afin d'exporter un objet 3D Sketchup pour Flash, il suffit de le publier au format KMZ et de modifier son extension .kmz en .zip. Le dossier compressé obtenu contient le fichier du modèle 3D au format DAE et un fichier XML de métadonnées. Seul le fichier DAE et les textures éventuellement associées sont requis.

Dans cette section, nous présentons comment récupérer un objet 3D de la banque d'objets de Google Sketchup pour le convertir du format natif Sketchup en DAE pour Flash et PaperVision.



Pour accéder directement à une bibliothèque d'objets 3D KMZ, sans utiliser Sketchup, reportez-vous à l'adresse : <http://sketchup.google.com/3dwarehouse/>.

Pour convertir des fichiers 3D natifs de Google dans un format compatible Flash, procédez comme suit :

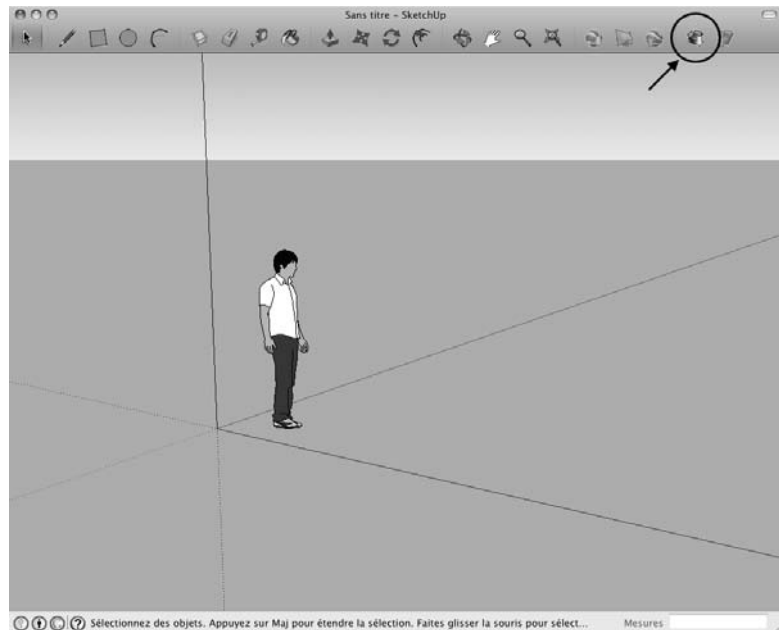
1. Téléchargez la version gratuite de Sketchup à : <http://sketchup.google.com/intl/fr/download/gsu.html>.
2. Validez toutes les étapes de l'installation.
3. Puis lancez l'application. Au démarrage, Sketchup affiche un écran avec différentes aides de grande qualité (voir Figure 10.19).
4. Cliquez directement sur Commencer à utiliser Sketchup. Une scène 3D s'ouvre (voir Figure 10.20).

Figure 10.19

Lancement
de Google
Sketchup 7.

**Figure 10.20**

Nouvelle scène
dans Sketchup.

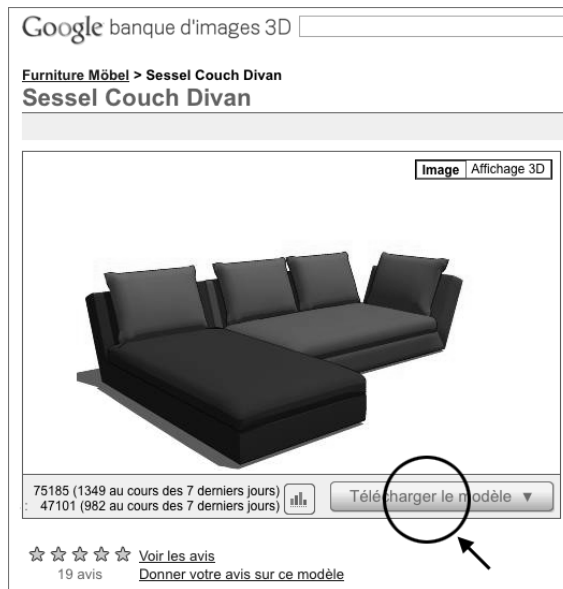


5. En haut et à droite de la fenêtre, cliquez sur le bouton Télécharger des modèles (voir Figure 10.20). Une fenêtre de navigateur s'ouvre sur la page de la banque d'objets 3D de Google.
6. Saisissez une requête. Par exemple, sofa ou Sessel Couch Divan. Puis validez. Vous accédez à une liste d'objets.

7. Sélectionnez celui de votre choix en le cliquant. Puis, dans la fiche descriptive de l'article, en bas de l'image de présentation, cliquez sur le lien Télécharger le modèle (voir Figure 10.21).

Figure 10.21

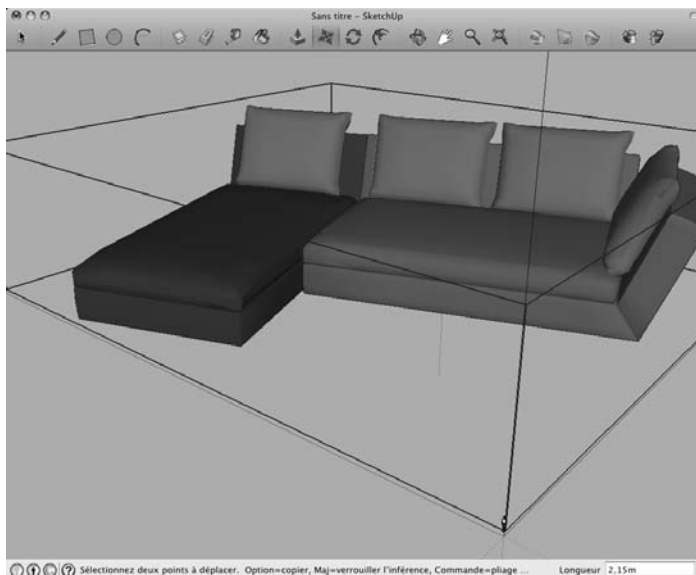
Télécharger le modèle.



8. Un message demande de confirmer le chargement du modèle dans le nouveau document Sketchup.
9. Confirmez et l'objet apparaît instantanément dans la scène. Vous pouvez l'exporter (voir Figure 10.22).

Figure 10.22

Modèle importé.



10. Si l'objet n'apparaît pas directement au centre de la scène, zoomez en arrière avec la roulette de la souris ou avec l'outil Loupe.
11. Repositionnez l'objet vers le point d'origine de la scène, à l'aide de l'outil de déplacement (Outil Déplacer/Copier en forme de 4 flèches cardinales rouges).
12. Supprimez le personnage de la scène en cliquant dessus et en faisant retour ou Suppr, au clavier.
13. Faites Fichier > Exporter > Modèle 3D. Nommez le document "sofa.kmz".
Une fenêtre affiche des informations sur les propriétés de l'objet.
14. Refermez la fenêtre pour continuer. Vous pouvez conserver éventuellement le document au format natif pour le modifier si nécessaire en faisant Fichier > Enregistrer, au format SKP.
15. Puis, quittez Sketchup.



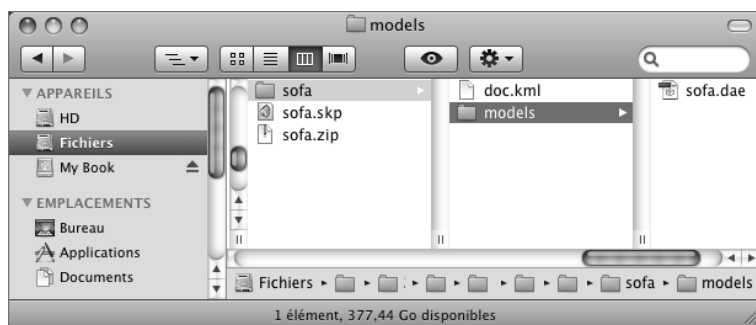
Si vous souhaitez pouvoir identifier des groupes d'objets pour PaperVision, en vue de modifier depuis PaperVision les teintes des composantes de forme des objets, utilisez la fenêtre Structure de Sketchup. Puis renommez chaque composante de l'objet 3D affiché dans cette liste. Aidez-vous éventuellement de l'option Créer un groupe, disponible par un clic-droit sur les formes graphiques composant les objets dans la scène.

Une fois le document exporté au format MKZ, vous pouvez en extraire le fichier Collada DAE.

Modifiez l'extension du fichier "sofa.kmz" en "sofa.zip". Confirmez éventuellement la boîte de dialogue d'avertissement qui apparaît. Puis, décompressez le fichier obtenu. Un dossier qui porte le même nom que le document apparaît. À l'intérieur, nous distinguons le fichier DAE isolé dans un répertoire nommé "models" (voir Figure 10.23).

Figure 10.23

Extraction du fichier DAE pour Flash et PaperVision.



Nous pouvons conserver la structure des sous-dossiers. Il n'en sera que plus simple pour gérer les éventuelles modifications par la suite. Le fichier DAE est maintenant exploitable directement dans Flash, en ActionScript.



Pour en savoir plus sur Sketchup, consultez cette adresse : <http://sketchup.google.com/intl/fr>. Pour accéder à une documentation détaillée, en texte et sous la forme de tutoriels vidéo sur la modélisation avec le logiciel Sketchup, consultez les adresses suivantes :

- <http://sketchup.google.com/intl/fr/training/videos.html>.
- <http://sketchup.google.com/support/>.

À retenir

- Vous pouvez exporter des objets 3D pour Flash et PaperVision si celui-ci est au format Collada DAE.
- Google Sketchup est une application 3D libre qui donne accès à des modèles 3D dans un format compatible avec Flash et PaperVision.
- Pour convertir un fichier KMZ en fichier DAE, il suffit de remplacer l'extension .kmz en .zip et décompresser le fichier obtenu.

Programmer les mouvements de caméra 3D

Au cours de cette section, nous utilisons un modèle d'objet 3D de référence Sessel Couch Divan, téléchargé depuis la banque d'objets 3D de Google. Nous allons voir comment ajouter dans le scénario de l'interface auteur de Flash, les commandes ActionScript 3D de PaperVision pour animer une caméra et l'objet 3D.

Dans cet exemple, nous utilisons des transitions de type TweenMax que nous appliquons à l'objet Camera en vue de réaliser plusieurs dispositifs de présentation.

Nous commençons par créer une animation autonome de la caméra 3D pour approcher l'objet et donner l'illusion que c'est lui qui s'approche de l'écran. Ensuite, nous associons ces actions à une occurrence de bouton pour effectuer un zoom de caméra, afin de permettre à l'utilisateur de contrôler lui-même le mouvement de la caméra. Nous ajoutons enfin un comportement à l'objet 3D, afin de le faire évoluer devant la caméra, une fois celle-ci fixée.

Pour cela, nous utilisons la classe PaperVision accessible depuis Flash.



Exemples > [ch10_3DPaperVision_1 fla](#)

Dans le document "ch10_3DPaperVision_1 fla", nous pouvons voir un bouton en bas et à droite, en forme de loupe, qui est associé à une action sur caméra. Au-dessous, un habillage gris matérialise la limite de la zone d'affichage réservée pour la 3D telle que définie en ActionScript. Une fois publié, le sofa part du fond de la scène et arrive progressivement au

premier plan, tout en tournant sur lui-même. En cliquant sur le bouton Loupe, la caméra se rapproche un peu plus encore à chaque clic (voir Figures 10.24 à 10.26).

Figure 10.24

Aperçu du document publié au début de l'animation.



Figure 10.25

Aperçu du document publié au milieu de l'animation.

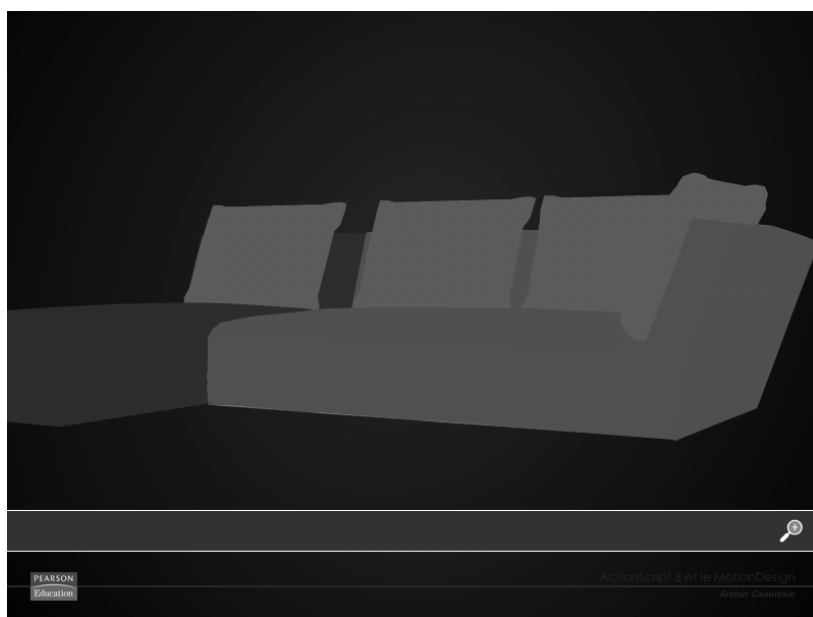
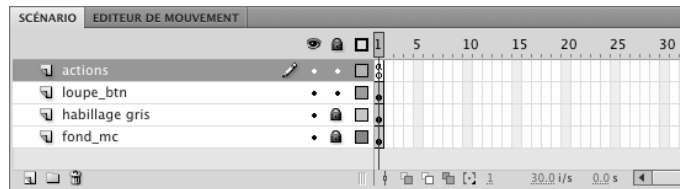


Figure 10.26

Aperçu du document publié à la fin de l'animation.

**Figure 10.27**

Scénario de la scène principale.



Dans la fenêtre de scénario de la scène principale, au-dessus du calque `fond_mc`, nous trouvons le bouton `loupe_btn`, l'habillage gris et le calque Actions (voir Figure 10.27). Dans le calque Actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.TweenMax;
import gs.easing.*;

import org.papervision3d.view.*;
import org.papervision3d.cameras.*;
import org.papervision3d.scenes.*;
import org.papervision3d.render.*;
import org.papervision3d.objects.primitives.*;
import org.papervision3d.events.*;
import org.papervision3d.materials.*;
import org.papervision3d.objects.parsers.DAE;

var espace:Viewport3D;
var rendu:BasicRenderEngine;
var scene:Scene3D;
var camera:Camera3D;

var objetDAE:DAE = new DAE();
```

```

objetDAE.load("objets3D/sofa/models/sofa.dae");

//----- actions
initialisation();
function initialisation() {
    espace=new Viewport3D(stage.stageWidth,stage.stageHeight-110);
    addChild(espace);
    rendu = new BasicRenderEngine();
    scene = new Scene3D();
    camera = new Camera3D();
    //
    objetDAE.rotationX=0;
    objetDAE.rotationY=-90;
    objetDAE.rotationZ=0;
    objetDAE.x=0;
    objetDAE.y=0;
    objetDAE.z=0;
    camera.y=15;
    //
    activerAffichage3D();
}
function activerAffichage3D() {
    scene.addChild(objetDAE);
    TweenMax.to(camera, 5, {z:camera.z+900, delay:0, ease:Strong.easeInOut});
    addEventListener(Event.ENTER_FRAME, enBoucle);
}
function enBoucle(evt:Event) {
    objetDAE.yaw(-1);
    rendu.renderScene(scene, camera, espace);
}

//modifier un objet 3D
loupe_btn.addEventListener(MouseEvent.CLICK,deplacerObjetDAE);
function deplacerObjetDAE(evt:MouseEvent) {
    TweenMax.to(camera, 5, {zoom:camera.zoom+100, delay:0,
    ➡ ease:Strong.easeOut});
}

```

Dans la première partie de ce code, nous commençons par importer l'ensemble des sous-classes de la classe PaperVision. Nous ciblons, pour chacune d'entre elles, le dossier "org", comme point de référence de cet ensemble. Du fait que nous avons intégré le chemin de classes dans les préférences de l'application, nous spécifions ici le dossier org situé dans le répertoire "ClassesPersos" que nous avons préalablement créé.

Les classes importées sont utilisées pour les transitions TweenMax :

```

//----- initialisation
import gs.TweenMax;
import gs.easing.*;

```

Puis, viennent celles de PaperVision :

- `import org.papervision3d.view.*;` : la classe view permet de gérer l'environnement 3D.

- `import org.papervision3d.cameras.*;` : la classe `cameras` permet de créer une caméra.
- `import org.papervision3d.scenes.*;` : la classe `scenes` permet de créer la scène que constitue l'espace 3D.
- `import org.papervision3d.render.*;` : la classe `render` calcule l'affichage 3D.
- `import org.papervision3d.objects.primitives.*;` : la classe `primitives`, non utilisée dans ce document, nous permettra de créer plus tard des formes primitives, telles que sphères, cylindres, cônes, plans, ou cubes.
- `import org.papervision3d.events.*;` : la classe `events` autorise l'interactivité avec les objets de la scène 3D.
- `import org.papervision3d.materials.*;` : la classe `materials` permet d'appliquer des textures à partir d'images à des formes 3D. Nous l'utilisons également plus loin dans ce chapitre.
- `import org.papervision3d.objects.parsers.DAE;` : enfin, la classe `DAE` autorise la gestion d'objets 3D importés au format DAE.

D'autres classes sont disponibles. Vous pouvez les identifier en parcourant le contenu du dossier `org`.

À la suite, nous définissons quelques variables qui caractérisent les types d'objets requis pour la construction de l'univers en trois dimensions :

```
var espace:Viewport3D;
var rendu:BasicRenderEngine;
var scene:Scene3D;
var camera:Camera3D;
```

`Viewport3D` définit la surface affichée à l'écran. Plus loin dans le code, cette variable reçoit en paramètre les dimensions de la zone d'affichage en pixels. L'objet `BasicRenderEngine` est le moteur de rendu. L'objet `Scene3D` constitue le volume à l'intérieur duquel nous plaçons tous les éléments en 3D. L'objet `Camera3D` enfin désigne l'œil du spectateur, la caméra.

Plus bas, nous créons un nouvel objet `DAE` qui charge le fichier 3D dans l'interface. Nous indiquons le chemin qui part du document Flash vers le fichier, à l'intérieur de la structure de dossier `KMZ` que nous avons décompressé :

```
var objetDAE:DAE = new DAE();
objetDAE.load("objets3D/sofa/models/sofa.dae");
```

La deuxième partie du code exécute l'interface 3D. Les actions sont rassemblées dans des fonctions afin de permettre une gestion de l'affichage contrôlée dans le temps.

```
//----- actions
initialisation();
function initialisation() {
    espace=new Viewport3D(stage.stageWidth,stage.stageHeight-110);
    addChild(espace);
    rendu = new BasicRenderEngine();
```

```
scene = new Scene3D();
camera = new Camera3D();
//
objetDAE.rotationX=0;
objetDAE.rotationY=-90;
objetDAE.rotationZ=0;
objetDAE.x=0;
objetDAE.y=0;
objetDAE.z=0;
camera.y=15;
//
activerAffichage3D();
}
```

La première instruction de la première fonction définit la zone d’affichage du rendu. Nous spécifions ici une largeur équivalente à la largeur de la scène du document Flash (`stage.stageWidth`) et une hauteur de 110 pixels de moins que la hauteur du document (`stage.stageHeight-110`). La hauteur est réduite afin de préserver la visibilité des contrôles de navigation que nous avons placés sur la scène (outil Loupe, entre autres). Gardez à l’esprit que plus votre surface de rendu est importante, plus les ressources de la machine client seront sollicitées.

La construction d’une interface 3D implique l’affichage de deux types d’objets. D’abord, nous affichons la zone de rendu avec ici l’objet `espace` dont nous venons de définir les dimensions. Plus loin, nous affichons les objets de la scène en 3D (DAE et primitives éventuelles). L’objet `espace` agit comme une fenêtre à travers laquelle on peut observer l’objet `scene` en 3D. L’objet `scene` ne définit en rien l’affichage, il n’est qu’un conteneur, pas un objet d’affichage.

Cette distinction est importante, car si la zone de rendu peut être gérée comme tout contenu depuis la liste d’affichage. La scène, elle, ne peut être invoquée directement, pour être redistribuée au-dessous d’un autre objet par exemple. Pour permettre de placer des symboles de la scène Flash au-dessus de la scène 3D, nous devons affecter le premier objet d’affichage, l’objet `espace` et non l’objet `scene`. Ainsi, si nous souhaitons que notre création 3D s’affiche à l’arrière-plan du bandeau gris du document Flash, par exemple, nous spécifions `addChildAt(espace,1)` au lieu de simplement `addChild(espace)`. Les objets 3D sont ajoutés à la scène avec également la méthode `addChild`, mais cela ne les inclut pas pour autant dans la liste d’affichage globale. De cette manière, l’objet `scene` induit une obstruction sémantique entre le contenu Flash et le contenu géré par PaperVision.



Il est également possible de placer un symbole par-dessus un objet 3D en conservant toutefois l’objet `espace` au sommet de la liste d’affichage. Pour cela, il suffit d’importer le symbole à ajouter en tant que texture dans un objet primitif de type `Plane`, par exemple, puis de gérer la position de cette primitive sur l’axe des Z avec un positionnement frontal par rapport à la caméra. Reportez-vous à la section “Interactivité avec les touches du clavier” pour en savoir plus sur la construction des primitives.

Une fois la zone d’affichage définie, nous l’ajoutons donc à la liste d’affichage classique avec `addChild(espace)`.

À la suite, nous matérialisons les objets pour lesquels nous avons créé les variables en amont, les variables :

```
rendu = new BasicRenderEngine();
scene = new Scene3D();
camera = new Camera3D();
```

Puis, nous modifions les propriétés des objets avant de les afficher, pour les caler dans l'environnement 3D :

```
objetDAE.rotationX=0;
objetDAE.rotationY=-90;
objetDAE.rotationZ=0;
objetDAE.x=0;
objetDAE.y=0;
objetDAE.z=0;
camera.y=15;
```

Nous ajustons les rotations de l'objet 3D afin qu'il apparaisse de face à la caméra (rotationY=-90). Puis, nous modifions légèrement la position verticale de la caméra pour recentrer également l'objet dans la scène (camera.y=15).

En dessous, nous poursuivons avec l'appel d'une fonction qui va activer l'affichage 3D des objets de la scène, et engager les animations des contenus alors prêts à être animés :

```
function activerAffichage3D() {
    scene.addChild(objetDAE);
    TweenMax.to(camera, 5, {z:camera.z+900, delay:0, ease:Strong.easeInOut});
    addEventListener(Event.ENTER_FRAME, enBoucle);
}
function enBoucle(evt:Event) {
    objetDAE.yaw(-1);
    rendu.renderScene(scene, camera, espace);
}
```

La deuxième instruction de cette fonction active une interpolation de type TweenMax qui anime la propriété z de l'objet camera de manière à créer un travelling jusqu'à atteindre l'objet DAE (camera.z+=900).

Plus bas, un écouteur, associé à un gestionnaire de type ENTER_FRAME, active le calcul du rendu de la scène 3D (rendu.renderScene(scene, camera, espace)). Les trois objets scene, camera et espace sont passés en paramètre.

Une dernière instruction ajoute un effet d'animation sur l'objet DAE. Cette instruction adopte une syntaxe propre à l'environnement PaperVision et équivaut à une rotation en Y d'un pas d'incrément de 1° à chaque image. La méthode employée est yaw et donne : objetDAE.yaw(-1). Nous abordons d'autres méthodes de ce type dans la section suivante.

Le programme s'achève sur l'ajout d'un comportement sur le bouton Loupe. En cliquant dessus, nous lançons une nouvelle interpolation qui anime la caméra avec une autre propriété : zoom.

```
//modifier un objet 3D
loupe_btn.addEventListener(MouseEvent.CLICK,deplacerObjetDAE);
function deplacerObjetDAE(evt:MouseEvent) {
    TweenMax.to(camera, 5, {zoom:camera.zoom+100, delay:0,
    ➤ ease:Strong.easeOut});
}
```

Les propriétés `zoom` et `z` se différencie par un déplacement physique de la caméra pour `z`, tandis que `zoom` ne fait qu'agrandir l'image. Ainsi, la propriété `zoom` permet de se rapprocher de l'objet 3D sans jamais le traverser, alors que `z` passerait à travers.



La sous-classe `Camera3D` est parfois aussi appelée `FreeCamera3D` (dans d'anciennes versions de `PaperVision`). Selon la version du *package* que vous utilisez, vérifiez que le nom de votre classe `Camera` présente dans le code `ActionScript` de votre document correspond bien au nom du fichier `.as` disponible dans les sous-dossiers de la classe `PaperVision`.

Il peut arriver, selon le poids du fichier, que Flash annule le rendu à la publication. Le calcul du rendu est alors peut-être trop long pour Flash. Pour corriger ce problème, allez dans les paramètres de publication, et sous le champ mot de passe, allongez la durée limite d'exécution du script avant laquelle Flash annule le rendu.

À retenir

- Il est possible d'animer facilement la caméra dans un environnement 3D pour créer des travellings, des zooms et des trajectoires, grâce à la combinaison de propriétés 3D et d'interpolations de type `TweenMax`.
- Le contenu d'une scène 3D n'est pas accessible directement depuis la liste d'affichage, mais son enveloppe oui. Il est donc possible de placer les animations 3D entre différents objets dans la scène du document Flash.
- Les commandes `ActionScript` de `PaperVision` peuvent être rédigées directement depuis la fenêtre de scénario.

Le `zoom` se distingue d'un déplacement en `Z`, en cela qu'il ne traverse jamais l'objet, au contraire de `Z`.

Programmer les mouvements d'objets 3D

Dans cette section, nous allons ajouter des animations qui permettent de manipuler directement les objets 3D de format DAE, importés dans Flash.



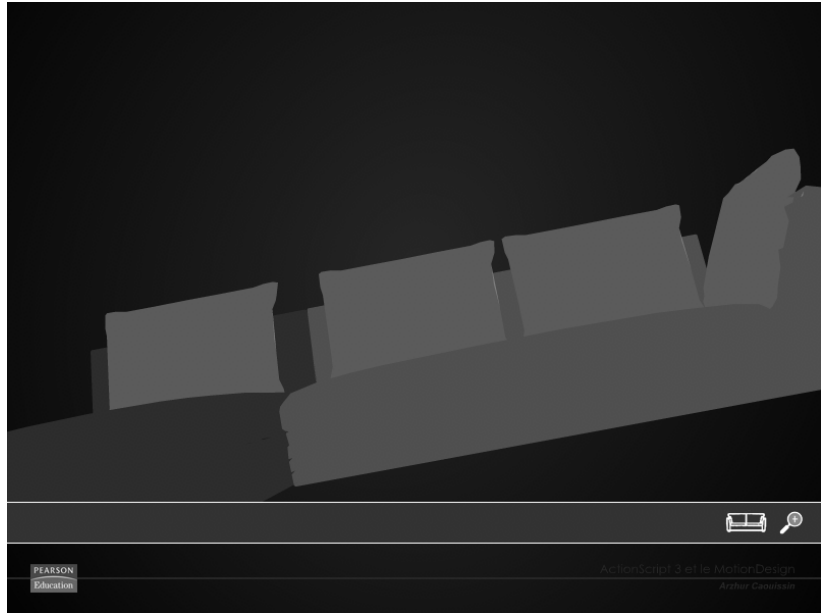
Exemples > `ch10_3DPaperVision_2 fla`

Le document "`ch10_3DPaperVision_2 fla`" est basé sur l'exemple précédent. Sur la scène principale, en plus de la bande grise et du bouton Loupe, figure un nouveau bouton nommé `sofa_btn` (voir Figure 10.28). Ce bouton est destiné à recevoir des instructions d'animation de l'objet 3D.

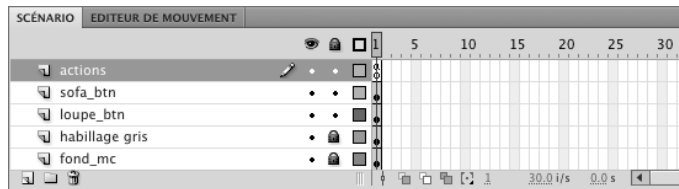
En publiant le document (voir Figure 10.29), lorsque le sofa est au premier plan, le bouton associé à l'objet le fait pivoter sur lui-même.

Figure 10.28

Aperçu du document publié.

**Figure 10.29**

Scénario de la scène principale.



Le calque actions affiche le code suivant :

```
//----- initialisation
import gs.TweenMax;
import gs.easing.*;

import org.papervision3d.view.*;
import org.papervision3d.cameras.*;
import org.papervision3d.scenes.*;
import org.papervision3d.render.*;
import org.papervision3d.objects.primitives.*;
import org.papervision3d.events.*;
import org.papervision3d.materials.*;
import org.papervision3d.objects.parsers.DAE;

var espace:Viewport3D;
var rendu:BasicRenderEngine;
var scene:Scene3D;
var camera:Camera3D;

var objetDAE:DAE = new DAE();
objetDAE.load("objets3D/sofa/models/sofa.dae");
```

```

//----- actions
initialisation();
function initialisation() {
    espace=new Viewport3D(stage.stageWidth,stage.stageHeight-110);
    addChild(espace);
    rendu = new BasicRenderEngine();
    scene = new Scene3D();
    camera = new Camera3D();
    //
    objetDAE.rotationX=0;
    objetDAE.rotationY=-90;
    objetDAE.rotationZ=0;
    objetDAE.x=0;
    objetDAE.y=0;
    objetDAE.z=0;
    camera.y=15;
    camera.rotationY=-10;
    //
    activerAffichage3D();
}

function activerAffichage3D() {
    scene.addChild(objetDAE);
    TweenMax.to(camera, 5, {z:camera.z+900, delay:0, ease:Strong.easeInOut});
    addEventListener(Event.ENTER_FRAME, enBoucle);
}
//
function enBoucle(evt:Event) {
    objetDAE.yaw(-1);
    objetDAE.roll(-1);
    objetDAE.pitch(-1);
    rendu.renderScene(scene, camera, espace);
}

//modifier un objet 3D
loupe_btn.addEventListener(MouseEvent.CLICK,deplacerObjetDAE);
function deplacerObjetDAE(evt:MouseEvent) {
    TweenMax.to(camera, 5, {zoom:camera.zoom+100, delay:0, ease:Strong.easeOut});
}

//Animer le sofa
sofa_btn.addEventListener(MouseEvent.CLICK,animerCamera);
function animerCamera(evt:MouseEvent) {
    TweenMax.to(objetDAE, 5, {rotationY:objetDAE.rotationY+45, rotationX:
    ➤ objetDAE.rotationX+45, rotationZ:objetDAE.rotationZ+45, delay:0,
    ➤ ease:Strong.easeInOut});
}

```

Dans cet exemple, nous animons l'objet DAE en le ciblant par le nom d'objet qui lui a été attribué (`objetDAE`), dès la déclaration des différents objets en début de code.

L'animation d'un objet 3D se construit de la même manière que l'animation d'une caméra. Les transitions `TweenMax` permettent de repositionner l'objet dans l'espace, de le faire pivoter en rotation sur les axes X, Y et Z.

Nous utilisons aussi d'autres méthodes, dont la méthode `yaw()`, déjà évoquée à la section précédente :

```
objetDAE.yaw(-1);
```

La méthode `yaw()` exécute une rotation latérale sur Y d'une valeur qui correspond au chiffre renseigné en paramètre. L'objet tourne au sol sur lui-même comme un tourniquet, de degré en degré :

```
objetDAE.roll(-1);
```

La méthode `roll()` exécute une rotation verticale sur X d'une valeur qui correspond au chiffre renseigné en paramètre. L'objet tourne ici sur lui-même, mais l'axe de rotation est parallèle à l'écran :

```
objetDAE.pitch(-1);
```

La méthode `pitch()` exécute une rotation frontale en Z d'une valeur correspondant au chiffre renseigné en paramètre. L'objet tourne sur lui-même, comme les aiguilles d'une montre.

Ces trois méthodes peuvent être appliquées indépendamment les unes des autres et cumulées à des rotations X, Y et Z. Dans notre exemple, les trois méthodes donnent un effet de rotation hybride qui fait évoluer l'objet dans toutes les directions simultanément. Vous pouvez les désactiver une par une pour observer leur comportement individuel.

En fin de programme, un écouteur est attaché au bouton `sofa_btn` et effectue une interpolation de type `TweenMax` en modifiant les paramètres de rotation X, Y et Z de l'objet DAE.



Optimiser un site 3D. Certains sites, bien que très spectaculaires, utilisent le même principe de navigation spatiale 3D avec des transitions de type `TweenMax` ou `Caurina`. Le site *System Bolaget Kampanj* (suédois) en est le parfait exemple. Il affiche au premier plan un village modélisé en 3D. À l'arrière-plan, sur la scène principale du document Flash, des panoramiques d'images PNG sont animés. Pour garantir la fluidité de l'effet 3D, les PNG du décor sont isolés dans des clips distincts et animés avec un coefficient d'accélération (voir Chapitre 1). La superposition de la scène 3D et 2D donne l'illusion d'un ensemble entièrement 3D. En isolant une partie du décor sous la forme d'images fixes, plutôt que de tout réaliser en 3D, nous optimisons considérablement le poids d'un projet. Vous pouvez consulter le site référent à l'adresse suivante : <http://www.systembolagetkampanj.se/forskar-rapport/>.

À retenir

- Il est possible d'animer les propriétés 3D d'un objet DAE importé dans le document Flash, pour le faire évoluer, se déplacer ou pivoter, sur action de l'utilisateur, par exemple.
- Des méthodes d'animation (`roll`, `yaw` et `pitch`) permettent de simplifier les mouvements de ces objets dans l'espace 3D et peuvent être cumulées avec l'animation d'autres propriétés.

Interactivité avec les touches du clavier

La représentation d'objets en volume implique la nécessité de contrôler la navigation dans l'espace. Dans cette section, nous allons voir comment reconstituer un décor virtuel à partir de formes primitives texturées, puis comment y naviguer à l'aide des commandes du clavier. Pour cela, nous devons déterminer la possibilité d'effectuer une combinaison de touches (Maj+Flèche droite, par exemple). Enfin, nous modifierons le sens de défilement des flèches du clavier, en fonction de l'orientation de la caméra dans le décor. Pour guider l'utilisateur, une boussole est placée sur la scène du document Flash et donne l'orientation de la caméra dans l'espace 3D.



Exemples > ch10_3DPaperVision_3 fla

Dans le document "ch10_3DPaperVision_3 fla", sur la scène principale, au-dessus du calque fond_mc et de l'habillage gris, apparaît une boussole. À l'intérieur du symbole qui le compose, deux clips sont répartis vers les calques et possèdent leur nom d'occurrence.

En publiant le document, des images tapissent des pans de murs virtuels. En activant les flèches du clavier, vous avancez, reculez ou vous déplacez à droite et à gauche. En appuyant simultanément sur majuscule et la touche flèche latérale, vous effectuez une rotation à 90° et pouvez reprendre une circulation vers l'avant en appuyant de nouveau sur la flèche du haut (voir Figures 10.30 et 10.31). L'aiguille de la boussole tourne en fonction de l'orientation définie pour la caméra.

Figure 10.30

Aperçu du document publié.

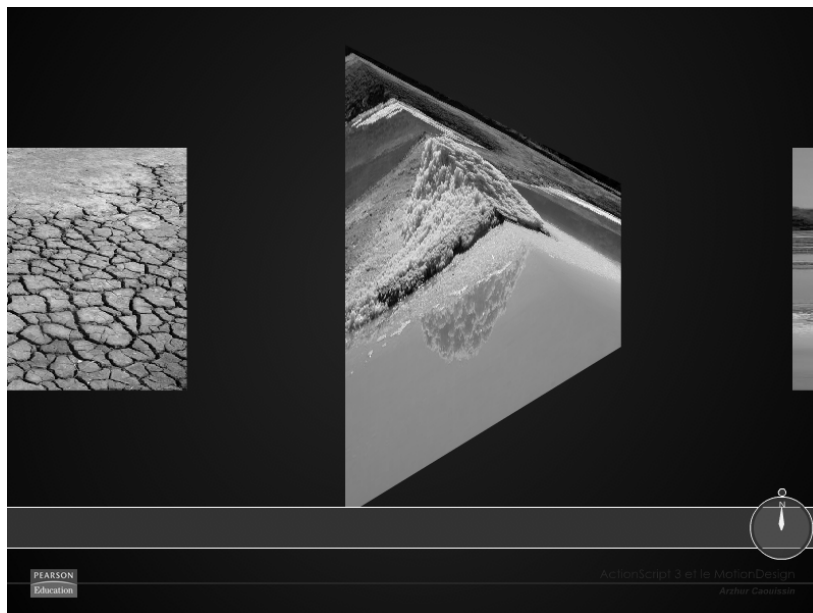


Figure 10.31

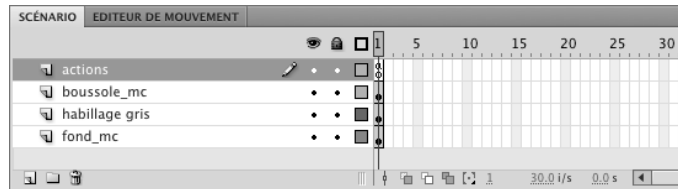
Aperçu du document publié, plus loin dans l'animation.



Dans la fenêtre de scénario, l'habillage gris et la boussole apparaissent (voir Figure 10.32).

Figure 10.32

Aperçu du scénario de la scène principale.



Dans le calque Actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.TweenMax;
import gs.easing.*;

import org.papervision3d.view.*;
import org.papervision3d.cameras.*;
import org.papervision3d.scenes.*;
import org.papervision3d.render.*;
import org.papervision3d.objects.primitives.*;
import org.papervision3d.events.*;
import org.papervision3d.materials.*;
import org.papervision3d.objects.parsers.DAE;

var espace:Viewport3D;
var rendu:BasicRenderEngine;
var scene:Scene3D;
var camera:Camera3D;

var plan0:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo0.jpg"),400,200);
```

```

var plan1:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo1.jpg"),400,200);
var plan2:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo2.jpg"),400,200);
var plan3:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo3.jpg"),400,200);
var plan4:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo4.jpg"),400,200);
var plan5:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo5.jpg"),400,200);

//----- actions
initialisation();
function initialisation() {
    activerPapervision();
    //
    camera.y=10;
    plan0.rotationY=0;
    plan0.x=0;
    plan1.rotationY=90;
    plan1.x=2000;
    plan2.rotationY=-45;
    plan2.x=500;
    plan3.rotationY=45;
    plan3.x=1500;
    plan4.rotationY=0;
    plan4.x=900;
    plan5.rotationY=90;
    plan5.x=3000;
    //
    activerAffichage3d();
}
//
function activerPapervision() {
    espace=new Viewport3D(stage.stageWidth,stage.stageHeight-110);
    addChild(espace);
    rendu = new BasicRenderEngine();
    scene = new Scene3D();
    camera = new Camera3D();
}
function activerAffichage3d() {
    scene.addChild(plan0);
    scene.addChild(plan1);
    scene.addChild(plan2);
    scene.addChild(plan3);
    scene.addChild(plan4);
    scene.addChild(plan5);
    addEventListener(Event.ENTER_FRAME, enBoucle);
    stage.addEventListener(KeyboardEvent.KEY_DOWN,navigationClavierDOWN);
    stage.addEventListener(KeyboardEvent.KEY_UP,navigationClavierUP);
}
//
function enBoucle(evt:Event) {
    rendu.renderScene(scene, camera, espace);
}

//----- contrôle clavier
var sens:Number=0;
var toucheBase:Number=0;
var toucheSupplementaire:Number=0;

```

```

//
function navigationClavierDOWN(evt:KeyboardEvent) {
    trace ("le code de la touche enfoncée est : "+evt.keyCode)
    //
    if (evt.keyCode==16) {
        toucheBase=16;
    }
}
function navigationClavierUP (evt:KeyboardEvent) {
    trace ("le code de la touche enfoncée est : "+evt.keyCode)
    trace ("sens = "+sens);
    trace("toucheBase = "+toucheBase)
    trace("toucheSupplementaire = "+toucheSupplementaire)
    //
    if (toucheBase==16 && toucheSupplementaire==37) {
        TweenMax.to(camera, 1, {rotationY:camera.rotationY-90, delay:0,
        ➡ ease:Strong.easeOut});
        toucheBase=999;
        toucheSupplementaire=999;
        sens-=90;
        if (sens<=-360) {
            sens=0;
        }
    }
    if (toucheBase==16 && toucheSupplementaire==39) {
        TweenMax.to(camera, 1, {rotationY:camera.rotationY+90, delay:0,
        ➡ ease:Strong.easeOut});
        toucheBase=999;
        toucheSupplementaire=999;
        sens+=90;
        if (sens>=360) {
            sens=0;
        }
    }
}
//
if (sens==0 || sens==360) {
    if (evt.keyCode==37) {
        toucheSupplementaire=37;
        TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==38) {
        toucheSupplementaire=38;
        TweenMax.to(camera, 1, {z:camera.z+200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==39) {
        toucheSupplementaire=39;
        TweenMax.to(camera, 1, {x:camera.x+200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==40) {
        toucheSupplementaire=40;
        TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
    }
}
//
if (sens==90 || sens==270) {
    if (evt.keyCode==37) {
        toucheSupplementaire=37;
        TweenMax.to(camera, 1, {z:camera.z+200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==38) {
        toucheSupplementaire=38;
    }
}

```

```

        TweenMax.to(camera, 1, {x:camera.x+200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==39) {
        toucheSupplementaire=39;
        TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==40) {
        toucheSupplementaire=40;
        TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
    }
}
//
if (sens== -90 || sens== -270) {
    if (evt.keyCode==37) {
        toucheSupplementaire=37;
        TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==38) {
        toucheSupplementaire=38;
        TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==39) {
        toucheSupplementaire=39;
        TweenMax.to(camera, 1, {z:camera.z+200, delay:0,
        ease:Strong.easeOut});
    }
    if (evt.keyCode==40) {
        toucheSupplementaire=40;
        TweenMax.to(camera, 1, {x:camera.x+200, delay:0,
        ease:Strong.easeOut});
    }
}
//
if (sens== 180 || sens== -180) {
    if (evt.keyCode==37) {
        toucheSupplementaire=37;
        TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==38) {
        toucheSupplementaire=38;
        TweenMax.to(camera, 1, {z:camera.z+200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==39) {
        toucheSupplementaire=39;
        TweenMax.to(camera, 1, {x:camera.x+200, delay:0, ease:Strong.easeOut});
    }
    if (evt.keyCode==40) {
        toucheSupplementaire=40;
        TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
    }
}
//
TweenMax.to(boussole_mc.fondBoussole_mc, 1, {rotation:-sens, delay:0,
➡ ease:Strong.easeOut});
}

```


Dans ce programme, basé sur les documents précédents, nous avons substitué l'objet DAE par des formes primitives de type `Plane` (plan) :

```
var plan0:Plane=new Plane(new
    BitmapFileMaterial("images/galerie/photo0.jpg"),400,200);
```

En paramètre de la méthode `new Plane()`, nous ciblons l'image que nous voulons afficher en tant que texture, puis, sa largeur et sa hauteur, qui sont exprimées en pixels.



Les primitives de PaperVision. Il existe plusieurs types de primitives disponibles dans PaperVision. Chaque objet peut recevoir une texture et posséder des dimensions libres si bien qu'une sphère ou un cylindre sont couramment employés pour créer des univers entiers (des ciels ou des panoramiques par exemple). Ainsi, pour créer une sphère de position `x=0` et `y=0`, par exemple, vous inscrivez :

```
var sphere:Sphere=new Sphere(new BitmapFileMaterial("images/ciel.jpg"),0,0);
```

Puis :

```
scene.addChild(sphere);
```

Vous pouvez éventuellement ajuster l'échelle des formes primitives avec la propriété `scale`.

Sur le même principe, la méthode `Plane()` génère une forme plate, un plan rectangulaire. `Cube()` désigne un cube (voir aussi http://forum.hardware.fr/hfr/Programmation/Divers-6/cube-site-entrees-sujet_125429_1.htm). `Cone()`, crée un cône et `Cylinder()` crée un cylindre. Pour le descriptif détaillé de construction de ces objets, avec des textures bitmaps, consultez l'adresse suivante : <http://astrois.info/blog/flash/realiser-une-visionneuse-de-panorama-vr-en-as3>.

Plus loin, nous gérons d'abord l'emplacement des objets :

```
plan0.rotationY=0;
plan0.x=0;
```

Puis, nous les ajoutons à la liste d'affichage :

```
scene.addChild(plan0);
```

Viennent ensuite les contrôles du clavier, activés par deux gestionnaires de la classe `KeyboardEvent` :

```
stage.addEventListener(KeyboardEvent.KEY_DOWN,navigationClavierDOWN);
stage.addEventListener(KeyboardEvent.KEY_UP,navigationClavierUP);
```

Nous utilisons deux gestionnaires, l'un pour détecter les touches enfoncées (`KEY_DOWN`) et le second pour les touches relevées (`KEY_UP`).

Puisque le contrôle de deux touches enfoncées simultanément n'est pas pris en compte dans une même structure conditionnelle, nous avons contourné ce problème en distinguant deux fonctions. Ainsi, la première fonction se charge de détecter une touche enfoncée pendant que la seconde détecte une autre touche, relevée. Deux touches appuyées, puis relâchées presque simultanément, peuvent donc être vérifiées.

Dans la première fonction, nous gérons la combinaison avec la touche Majuscule (`keyCode=16`) et dans la seconde fonction, les touches flèches (`37` = flèche gauche, `38` = flèche haut, `39` = flèche droite et `40` = flèche bas).

La première fonction commence par identifier la touche enfoncée en affichant son nom. Cela permet d'évaluer, en fonction du système et du clavier, si les valeurs utilisées correspondent bien à l'environnement de développement.

```
trace ("le code de la touche enfoncée est : "+evt.keyCode)
```

Cette fonction définit ensuite une variable qui nous renseigne sur le fait que la touche Majuscule (keyCode==16) est enfoncée :

```
if (evt.keyCode==16) {
    toucheBase=16;
}
```

Les touches de commande

En ActionScript 3, les touches de commande sont également identifiables par des noms de propriétés :

- La propriété `shiftKey` correspond à la touche Shift ou Majuscule.
- La propriété `altKey` correspond à la touche Alt.
- La propriété `ctrlKey` correspond à la touche Contrôle.

Le langage

Cette valeur est initialisée dans la fonction suivante (`navigationClavierUP`), lorsque l'utilisateur relâche les touches du clavier. En d'autres termes, les actions associées à la touche Majuscule n'ont d'effet que si celle-ci reste enfoncée pendant que la deuxième fonction est exécutée. Seule la combinaison des deux touches modifie l'action à l'exécution de la deuxième fonction.

La deuxième fonction vérifie la touche complémentaire lorsqu'elle est relâchée en nous informant d'abord, dans la fenêtre de sortie, sur son nom, ainsi que différentes valeurs utilisées dans les autres conditions :

```
trace ("le code de la touche enfoncée est : "+evt.keyCode)
trace ("sens = "+sens);
trace("toucheBase = "+toucheBase)
trace("toucheSupplementaire = "+toucheSupplementaire)
```

Les deux conditions suivantes appliquent, si la touche Majuscule est effectivement enfoncée, une rotation en Y de 90°, dans un sens ou dans l'autre :

```
if (toucheBase==16 && toucheSupplementaire==37) {
    TweenMax.to(camera, 1, {rotationY:camera.rotationY-90, delay:0,
    ➤ ease:Strong.easeOut});
    toucheBase=999;
    toucheSupplementaire=999;
    sens=-90;
    if (sens<=-360) {
        sens=0;
    }
}
if (toucheBase==16 && toucheSupplementaire==39) {
    TweenMax.to(camera, 1, {rotationY:camera.rotationY+90, delay:0,
    ➤ ease:Strong.easeOut});
```

```

    toucheBase=999;
    toucheSupplementaire=999;
    sens+=90;
    if (sens>=360) {
        sens=0;
    }
}

```

La valeur affectée à la touche Majuscule (`toucheBase=16`) est initialisée dès que l'action est activée. Ainsi, l'utilisateur peut effectuer d'autres commandes, qui n'impliquent pas la touche Majuscule, sans risquer de compromettre les autres actions du programme. Pour être initialisée, la valeur est passée à 999, qui ne correspond à aucune touche.

Si, en plus, la touche relâchée correspond à une des quatre flèches du clavier (`&& toucheSupplementaire==37`), alors, le sens de la caméra est modifié :

```

    if (toucheBase==16 && toucheSupplementaire==37) {
        TweenMax.to(camera, 1, {rotationY:camera.rotationY-90, delay:0,
        ➤ ease:Strong.easeOut});
        toucheBase=999;
        toucheSupplementaire=999;
        sens-=90;
        if (sens<=-360) {
            sens=0;
        }
    }
}

```

L'action est déclinée aussi pour la flèche droite (39). Plus bas, des conditions lancent les différents mouvements de caméra, sans rotation, pour les combinaisons de touche simples. Ces animations sont exécutées même si l'utilisateur a enfoncé la touche Majuscule :

```

    if (sens==0 || sens==360) {
        if (evt.keyCode==37) {
            toucheSupplementaire=37;
            TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
        }
        //actions des autres flèches du clavier
    }
}

```

Le principe est décliné pour chaque touche (37, 38, 39 ou 40) et, surtout, selon chaque valeur d'orientation de caméra définie par `sens`.

Selon la position relative de la caméra, il faut en effet ajuster les commandes de sorte qu'un utilisateur qui a effectué une rotation de 90° à droite (Maj+Flèche droite) puisse continuer d'avancer en appuyant sur la flèche du haut, et non en appuyant sur la flèche de droite.

Voici plus précisément le mécanisme qui régit l'ensemble des instructions à exécuter : dans un espace 3D, au sol, sont imprimés les axes X et Z. Pour avancer tout droit, nous incrémen-
tons notre position sur l'axe Z. Pour nous déplacer latéralement, en crabe, nous utilisons donc l'axe X. Si maintenant nous faisons pivoter la caméra de 90° à droite, nous percevons désormais ce qui longe l'axe X, anciennement à notre droite, en face de nous. Mais, la commande du clavier Flèche haut, elle, continue de contrôler le déplacement sur Z. Si nous acti-
vons la touche Flèche haut, nous nous déplaçons donc maintenant sur la gauche (axe Z). Pour que le déplacement à gauche se fasse désormais avec la flèche de gauche, nous devons définir une condition. Si nous voulons autoriser l'utilisateur à s'orienter indifféremment à droite ou à gauche et d'avant en arrière, nous devons définir quatre blocs de conditions.

Ces conditions vérifient l’orientation appliquée à la caméra à l’aide de la variable sens. Les valeurs sont incrémentées ou diminuées de 90 à chaque modification de la rotation de la caméra en Y. Lorsque cette valeur atteint la limite de +360 ou –360, nous l’initialisons à 0. À l’intérieur de chaque condition, nous ajoutons donc des contrôles qui déterminent le type de déplacement à effectuer selon l’angle observé. Nous ajustons pour cela deux paramètres : le signe plus ou moins attribué au pas d’incréméntation ainsi que l’axe de défilement X ou Z :

```
TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
```

Pour terminer, une interpolation indique à l’aiguille de la boussole de pivoter en fonction de cette orientation :

```
TweenMax.to(boussole_mc.fondBoussole_mc, 1, {rotation:-sens, delay:0,
➡ ease:Strong.easeOut});
```



Vérifier les valeurs de ses propres touches clavier. Pour vérifier instantanément les valeurs de l’ensemble de vos touches clavier, placez le code suivant dans la fenêtre Actions d’un document Flash et publiez-le. Le document “ch10_3DpaperVision_3b fla” contient ce programme :

```
// afficher les commandes de clavier
for (var i:int = 33; i< 126;i++){
    trace(i + "\t: " + String.fromCharCode(i))
}
```

Dans les deux tableaux qui suivent, nous indiquons les valeurs numériques qui correspon- dent aux claviers ASCII français et américain. Vous y trouverez les valeurs vous permettant de développer d’autres actions à partir des commandes du clavier, selon sa configuration.

Tableau 10.1 : Valeurs des principales commandes de clavier ASCII

Dénomination des touches	Valeur numérique
Retour	8
Tabulation	9
Entrée	13
Majuscule	16
Ctrl/Contrôle	17
Alt	18
Pause/Break/Attn	19
Verrouillage majuscule	20
Escape	27
Page haut	33
Page bas	34
Fin	35
Sommaire (flèche haut et gauche, au-dessus de la touche fin)	36
Flèche gauche	37
Flèche haut	38
Flèche droite	39

Tableau 10.1 : Valeurs des principales commandes de clavier ASCII (suite)

Dénomination des touches	Valeur numérique
Flèche bas	40
Insert	45
Supprimer	46
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
a	65
b	66
c	67
d	68
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89
z	90
Touche Windows de gauche	91

Tableau 10.1 : Valeurs des principales commandes de clavier ASCII (suite)

Dénomination des touches	Valeur numérique
Touche Windows de droite	92
0 (pavé numérique)	96
1 (pavé numérique)	97
2 (pavé numérique)	98
3 (pavé numérique)	99
4 (pavé numérique)	100
5 (pavé numérique)	101
6 (pavé numérique)	102
7 (pavé numérique)	103
8 (pavé numérique)	104
9 (pavé numérique)	105
Multiplier	106
Additionner	107
Soustraire	109
Point décimal	110
Diviser	111
f1	112
f2	113
f3	114
f4	115
f5	116
f6	117
f7	118
f8	119
f9	120
f10	121
f11	122
f12	123
Verrouillage pavé numérique	144
Verrouillage défilement	145
Point-virgule	186
Égal	187
Commande	188
Tiret	189
Point	190
Slash	191
Accent grave	192
Crochet ouvert	219
Back slash	220
Crochet fermé	221
Simple quote ou apostrophe simple	222

Tableau 10. 2 : Valeur des principales commandes de clavier US

Dénomination des touches	Valeur numérique
Alt	18
Arrow down	40
Arrow left	37
Arrow right	39
Arrow up	38
Backspace	8
Caps Lock	20
Ctrl	17
Delete	46
End	35
Enter	13
Esc	27
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
Home	36
Insert	45
Num Lock	144
(NumPad) -	109
(NumPad) *	106
(NumPad) .	110
(NumPad) /	111
(NumPad) +	107
(NumPad) 0	96
(NumPad) 1	97
(NumPad) 2	98
(NumPad) 3	99
(NumPad) 4	100
(NumPad) 5	101
(NumPad) 6	102
(NumPad) 7	103

Tableau 10. 2 : Valeur des principales commandes de clavier US (suite)

Dénomination des touches	Valeur numérique
(NumPad) 8	104
(NumPad) 9	105
Page Down	34
Page Up	33
Pause	19
Print Scrn	44
Scroll Lock	145
Shift	16
Spacebar	32
Tab	9
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
1	49
2	50

Tableau 10. 2 : Valeur des principales commandes de clavier US (suite)

Dénomination des touches	Valeur numérique
3	51
4	52
5	53
6	54
7	55
8	56
9	57
0	48
`	222
-	189
,	188
.	190
/	191
;	186
[	219
\	220
]	221
`	192
=	187

Pour connaître la liste des principales commandes clavier supportées par le système Linux, reportez-vous à l'adresse suivante : http://www.comptechdoc.org/os/linux/howlinuxworks/linux_hlkeycodes.html.



Interactivité sur les objets 3D. Pour en savoir plus sur les interactions possibles avec les objets 3D, consultez la page suivante : <http://papervision3d-fr.com/2009/09/16/interactive-displayobject3d/>.

Gestion des lumières dans PaperVision. Pour en savoir plus sur l'ajout de lumières dans la scène 3D, consultez la page : <http://papervision3d-fr.com/2009/09/22/la-lumiere-dans-papervision3d/>.

Guide de référence PaperVision. Un guide de référence sur PaperVision est disponible à cette adresse : <http://papervision3d.googlecode.com/svn/trunk/as3/trunk/docs/class-summary.html>.

Aide en ligne de PaperVision. De nombreux utilisateurs de la classe PaperVision travaillent en mode de programmation objet en externalisant leur code sous la forme de classes et de packages dans des fichiers ayant l'extension .as. Des sites communautaires et des blogs font référence à cette méthode. Vous y trouverez des aides complémentaires pour vous guider sur les techniques avancées de programmation avec PaperVision :

- <http://content.madvertices.com/articles/PV3DTraining/default.htm#setUp>.
- http://wiki.mediabox.fr/tutoriaux/flash/papervision_3D_bases.
- <http://blog.pixlp.com/index.php/>.

À retenir

- Les positions 3D des objets et des caméras peuvent être contrôlées par les touches du clavier à l'aide de la propriété `keyCode`.
- Pour construire une navigation viable, vous devez détecter la rotation de la caméra et en déterminer l'axe de défilement `x` ou `z`.

Synthèse

Dans ce chapitre, vous avez appris à intégrer des objets 3D au sein de l'environnement Flash, à l'aide de la classe `PaperVision`, en programmant les commandes `PaperVision` directement dans le scénario. Vous savez placer les scènes 3D dans un décor multicalques et interagir avec les trajectoires animées de la caméra et déplacer les objets 3D, y compris en utilisant les touches du clavier.

Les bogues d'affichage, le poids limité, comparé à la qualité de résolution disponible en flux vidéo F4V favorise cependant encore largement la vidéo interactive à la technologie 3D interactive sur le Web aujourd'hui. Mais les innovations en cours en font un outil à ne pas négliger dans les développements à venir.

11

API d'affichage et de colorimétrie

Introduction

Bien que Flash intègre déjà tout type de données, il peut être intéressant de développer des fonctionnalités avancées sur des contenus traités dynamiquement. Par exemple, vous pouvez proposer une fonctionnalité de grossissement d'une image, une fonctionnalité de retouche colorimétrique d'une image, pour tester des ambiances et offrir de nouveaux services à des utilisateurs exigeants.

Flash propose un moteur de traitement d'images avancé. Il dispose, pour cela, de deux classes qui permettent de modifier intrinsèquement les pixels d'une image (`ColorMatrixFilter`) et de déformer un objet graphique (`Matrix`).

Dans ce chapitre, nous abordons le traitement des pixels de l'image à travers ces différentes méthodes. Nous voyons comment réaliser, entre autres, des interfaces sans crénelage, une loupe, des filtres de traitement colorimétrique, des fonctionnalités d'impression sur des zones définies de l'interface. Nous verrons ensuite, dans le prochain chapitre, comment exploiter ces techniques pour mettre au point, par exemple, un moteur de rendu en relief.

Lissage des images bitmap

Le lissage des images est requis dès que l'image suit une transformation d'échelle ou de rotation. Sans lissage, l'image animée apparaît crénelée et un voile cristallin semble même la déformer dès qu'elle subit un mouvement.

Comme évoqué aux Chapitres 5 et 9, nous avons vu comment lisser une image en activant l'option Autoriser le lissage, depuis les propriétés de chaque image, dans la bibliothèque. Mais, dans de nombreux cas, pour une galerie d'images dynamique par exemple, il peut être intéressant d'activer cette option avec ActionScript. Pour ce faire, nous utilisons la propriété `smoothing`.

Pour appliquer cette propriété, nous devons au préalable comprendre qu'elle s'applique seulement à un objet image, et non à son conteneur. Il en résulte que, pour lisser un `MovieClip`, nous devons, au préalable, créer une enveloppe image (`Bitmap`) qui capture ce que représente le `MovieClip`, pour seulement ensuite le lisser par ActionScript.

Dans cette section, nous allons voir comment lisser une image chargée dynamiquement sur la scène, dans un `MovieClip`.



Exemples > ch11_apiGraphisme_1.fla

Dans le document "ch11_apiGraphisme_1.fla", sur la scène principale, est positionné un MovieClip vide, nommé contenu_mc, ainsi qu'un bouton Loupe. Le bouton Loupe agrandit et effectue une rotation sur ce MovieClip, si bien que sans les propriétés de lissage, nous pouvons constater l'apparition d'un crénelage.

En publiant le document, l'image est d'abord chargée dans le MovieClip et affiche déjà un lissage. En activant le bouton Loupe, l'image effectue une rotation et subit un agrandissement, mais les pixels restent lisses, même à grande échelle (voir Figure 11.1).

Figure 11.1

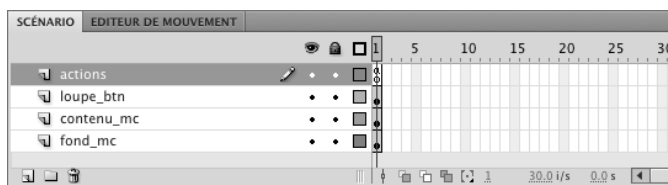
Aperçu du document publié.



Dans la fenêtre de scénario de la scène principale, au-dessus du calque fond_mc, figurent le symbole contenu_mc (vide) et le bouton loupe_btn (voir Figure 11.2).

Figure 11.2

Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
```

```

import gs.events.*;

//----- actions
// chargement
var chargeur = new Loader();
chargeur.load(new URLRequest("images/provence/Roussillon.jpg"));
chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementComplet);

function chargementComplet(evt:Event) {
    // lissage
    var imageDepart:Bitmap = evt.target.content;
    var informationImage:BitmapData=new
BitmapData(imageDepart.width,imageDepart.height);
    informationImage.draw(contenu_mc);
    imageDepart.smoothing=true;
    // affichage
    contenu_mc.addChild(imageDepart);
    contenu_mc.x=100;
    contenu_mc.y=30;
}

// bouton loupe
loupe_btn.addEventListener(MouseEvent.CLICK,tournerImage);
function tournerImage (evt:MouseEvent) {
    if (contenu_mc.rotation>-89) {
        TweenMax.to(contenu_mc, 2, {rotation:-90, scaleX:2, scaleY:2, x:0,
        ➡ y:520, delay:0, ease:Strong.easeInOut});
    } else {
        TweenMax.to(contenu_mc, 2, {rotation:0, scaleX:1, scaleY:1, x:100, y:30,
        ➡ delay:0, ease:Strong.easeInOut});
    }
}

```

Ce code est structuré en trois parties. La première appelle les classes utilisées pour les interpolations TweenMax. La deuxième charge l’image et la lisse. La troisième partie gère l’interactivité avec le bouton Loupe.

La première action de la deuxième partie charge l’image, comme vu au Chapitre 5. Aussitôt après, la fonction appelée affiche l’organisation de l’image chargée, pour lui permettre d’obtenir un lissage :

```

function chargementComplet(evt:Event) {
    // lissage
    var imageDepart:Bitmap = evt.target.content;
    var informationImage:BitmapData=new
BitmapData(imageDepart.width,imageDepart.height);
    informationImage.draw(contenu_mc);
    imageDepart.smoothing=true;
    // affichage
    contenu_mc.addChild(imageDepart);
    contenu_mc.x=100;
    contenu_mc.y=30;
}

```

Pour appliquer un lissage, nous définissons d'abord un objet vectoriel texturé d'une image. Et seulement une fois cet objet matérialisé, nous pouvons lui appliquer un lissage.

Pour lisser une image chargée dynamiquement, nous procédons donc en plusieurs étapes. D'abord, nous commençons par définir le flux chargé (evt.target.content) comme un objet image avec un type Bitmap :

```
var imageDepart:Bitmap = evt.target.content;
```

Puis, nous plaçons les données de l'image (ses informations colorimétriques), dans un nouvel objet BitmapData :

```
var informationImage:BitmapData=new  
    ↳ BitmapData(imageDepart.width,imageDepart.height);
```

Nous indiquons, en paramètre de cette méthode, les dimensions de cet objet (largeur et hauteur). Les largeur et hauteur indiquées sont ici celles de l'objet Bitmap déjà identifié.

Les données véhiculées par l'objet BitmapData sont matérialisées sur le symbole contenu_mc par l'instruction suivante :

```
informationImage.draw(contenu_mc);
```

L'image est en fait placée sur l'enveloppe extérieure du MovieClip contenu_mc et non à l'intérieur. Pour le vérifier, nous pouvons lire le contenu du symbole en utilisant l'une des instructions suivantes. Ces instructions retournent chacune une valeur nulle. L'image n'est donc pas contenue dans l'objet, mais bien en surface :

```
trace(contenu_mc.numChildren) ; // 0  
trace(contenu_mc.getChildAt(0)) ; // Index hors limites
```

Par contre, c'est bien à l'objet vectoriel texturé BitmapData que nous appliquons la propriété de lissage smoothing :

```
imageDepart.smoothing=true;
```

Nous pouvons alors afficher l'objet, le placer avec des propriétés x et y, voire, le déformer. Il restera lisse :

```
contenu_mc.addChild(imageDepart);  
contenu_mc.x=100;  
contenu_mc.y=30;
```

Le code se termine sur la gestion du bouton qui effectue les interpolations d'échelle et de rotation :

```
// bouton loupe  
loupe_btn.addEventListener(MouseEvent.CLICK,tournerImage);  
function tournerImage (evt:MouseEvent) {  
    if (contenu_mc.rotation>-89) {  
        TweenMax.to(contenu_mc, 2, {rotation:-90, scaleX:2, scaleY:2, x:0,  
            ↳ y:520, delay:0, ease:Strong.easeInOut});  
    } else {  
        TweenMax.to(contenu_mc, 2, {rotation:0, scaleX:1, scaleY:1, x:100, y:30,  
            ↳ delay:0, ease:Strong.easeInOut});  
    }  
}
```

Dans la structure conditionnelle, si le conteneur affiche une rotation inférieure à -89°, la première transition est jouée et porte ce conteneur à une rotation de 90°. Lorsque la condition

n’est plus vérifiée, une interpolation inverse est lancée et retourne l’objet à une rotation 0, supérieure donc à -89° . Chaque interpolation, en plus de faire tourner l’objet, applique une modification d’échelle. Sans le lissage, l’image laisserait apparaître un crénelage. Pour vérifier que le lissage est bien actif, vous pouvez neutraliser l’instruction suivante, en commentaire, puis la restaurer :

```
//imageDepart.smoothing=true;
```

À la Figure 11.3, nous soulignons la différence entre l’image affichée avec un lissage et celle qui bénéficie du lissage. La différence est surtout perceptible pendant le mouvement, où l’image saute, en plus d’apparaître crénelée si elle n’est pas lissée.

Figure 11.3

Comparaison des deux images.



Le document actuel traite une image chargée dynamiquement. Une déclinaison de cet exemple, adapté pour une image placée manuellement dans le scénario, est disponible dans le fichier "ch11_apiGraphisme_1b fla".

À retenir

- Pour lisser une image, nous devons la placer en tant qu’objet Bitmap sur un conteneur et lui passer la propriété `smoothing` sur `true`.
- Le lissage augmente les ressources graphiques nécessaires pour l’exécution du programme. Il ne faut pas en abuser.

Lissage des graphismes vectoriels

Tout comme pour les images bitmap, il est possible d’optimiser aussi l’affichage des formes graphiques vectorielles. Pour cela, nous utilisons la propriété `cacheAsBitmap` que nous définissons sur `true`.

Dans cette section, nous appliquons cette propriété dynamiquement sur un symbole qui contient plusieurs occurrences d’une forme graphique, importée d’Illustrator.



Exemples > ch11_apiGraphisme_2.fla

Dans le document "ch11_apiGraphisme_2.fla", sur la scène principale, apparaît un MovieClip. À l'intérieur, plusieurs occurrences d'une plume vectorielle sont superposées (voir Figure 11.4).

Figure 11.4

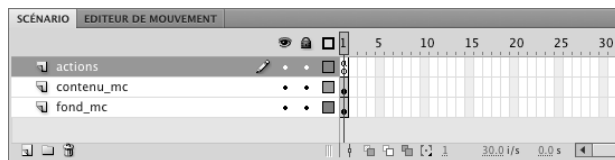
Aperçu du document.



Dans le scénario, seul le symbole contenu_mc apparaît entre le calque fond_mc et le calque actions (voir Figure 11.5).

Figure 11.5

Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, une instruction applique la mise en cache sur le symbole conteneur contenu_mc :

```
//----- Mise en cache
contenu_mc.cacheAsBitmap=true;
```

La mise en cache peut également être appliquée directement depuis l'Inspecteur de propriétés, en cliquant sur l'option intitulée Cache sous forme de bitmap (voir Figure 11.6).

Chaque mise en cache induit la création d'une image bitmap à partir des éléments vectoriels affichés dans un clip. En utilisant cette option, nous alourdissons considérablement le document. Quelques restrictions d'utilisation doivent donc être soulevées :

- Limitez le nombre de symboles qui utilisent cette propriété. Préférez rassembler les formes graphiques vectorielles dans un seul et unique symbole.

- N’animez pas le symbole dont le contenu est mis en cache (pas de rotation, pas de mise à l’échelle). Les déplacements simples sont possibles (animation par translation en X ou Y, ou déplacements avec `startDrag`). Car une déformation de l’image oblige le moteur de rendu à créer une nouvelle image à chaque modification. Vous ralentissez donc indéniablement les performances de l’application.
- Si l’image véhiculée doit apparaître à de nombreuses reprises, ou être superposée par des contenus déjà très gourmands en ressources, il est souhaitable d’activer cette option.
- Le texte contenu dans une zone défilante (avec un ascenseur par exemple) peut être mis en cache.
- Une illustration en arrière-plan d’un site, une carte du monde fixe et non zoomée, peut également être mise en cache.

Figure 11.6

Aperçu du document.



Vous pouvez lisser les bords d’une forme vectorielle en activant, directement depuis l’Inspecteur de propriétés, et sur la forme elle-même, dans la catégorie Remplissage et trait, l’option Repères située à droite du champ Échelle. Cette option aligne automatiquement les formes graphiques sur des abscisses et des ordonnées entières. Elle évite donc l’affichage parfois bancal de certains contours.

À retenir

- Il est possible d’optimiser les performances d’affichage pour les formes vectorielles. Nous utilisons pour cela la propriété `cacheAsBitmap`.
- La mise en cache des formes vectorielles ne doit cependant pas être systématique et ne doit concerner que les images non déformées, et non animées.

Effet loupe avec optique déformante

Pour créer une loupe, *a priori*, rien de plus simple que de dupliquer une zone de l'image à agrandir, dans un conteneur masqué. Mais pour que l'élément agrandi suive correctement le mouvement de déplacement de la loupe, nous devons définir une équation. De même, pour que la loupe affiche une image agrandie et déformée, nous devons aussi appliquer un flou sur le masque qui la canalise. Or, un masque ne supporte un flou que si celui-ci a été créé dynamiquement.

Dans cette section, nous utilisons une image placée dans le scénario pour créer un effet de loupe sur une lentille, mobile, par simple déplacement (voir Figure 11.7).

Figure 11.7

Aperçu du document publié.

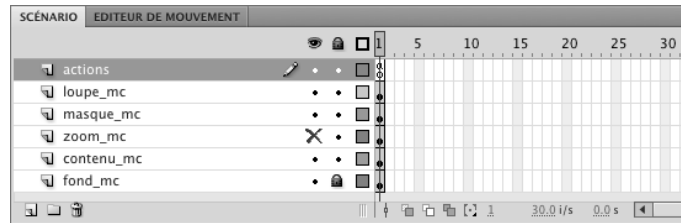


Exemples > ch11_apiGraphisme_3 fla et Exemples > ch11_apiGraphisme_3b fla

Dans le document "ch11_apiGraphisme_3 fla", sur la scène principale, figure le MovieClip contenu_mc, qui affiche une image. Les dimensions de ce MovieClip sont réduites à 50 %. Au-dessus, est placé un symbole nommé loupe_mc, constitué d'une forme dégradée semi-transparente et d'un bouton de déplacement. Sur la scène principale, un disque noir est placé hors champ, sur la droite. Ce MovieClip, nommé masque_mc, sert de masque pour l'image agrandie. L'image agrandie, elle, apparaît dans le scénario sous le nom d'occurrence zoom_mc (voir Figure 11.8). Son calque est désactivé mais demeure visible à la publication. Ce symbole est en fait une deuxième occurrence du même symbole contenu_mc, mais affiché à 100 %. L'image qu'il véhicule n'est donc pas déformée.

Figure 11.8

Aperçu du scénario.



Dans la fenêtre de scénario, au-dessus des autres calques, apparaît le calque actions. Il affiche le code suivant :

```
//----- initialisation
import flash.filters.BlurFilter;

loupe_mc.cacheAsBitmap=true;
var flou:BlurFilter=new BlurFilter(10,10,3);
masque_mc.filters=[flou];
zoom_mc.mask=masque_mc;
var zoneDeplacement:Rectangle=new Rectangle(150,100,470,350);

//----- actions
// déplacement de la loupe
loupe_mc.deplacerLoupe_btn.addEventListener(MouseEvent.CLICK, deplacerLoupe );
function deplacerLoupe(evt:MouseEvent) {
    loupe_mc.startDrag(false,zoneDeplacement);
}
addEventListener(MouseEvent.CLICK, arreterDeplacement );
function arreterDeplacement(evt:MouseEvent) {
    stopDrag();
}
// positionnement de l'image zoomée
addEventListener(Event.ENTER_FRAME,positionMasque);
function positionMasque(evt:Event) {
    masque_mc.x=loupe_mc.x;
    masque_mc.y=loupe_mc.y;
    zoom_mc.x=( ( loupe_mc.x*-1)+(stage.stageWidth/2)) + (zoom_mc.width/2) )
    ➡ -loupe_mc.width;
    zoom_mc.y=( ( loupe_mc.y*-1)+(stage.stageHeight/2)) + (zoom_mc.height/2) )
    ➡ -loupe_mc.height;
}
```

Le programme est organisé en trois parties : l’initialisation, les actions de déplacement de la loupe et la gestion de l’image agrandie.

La loupe est constituée d’un MovieClip qui contient principalement une forme graphique presque transparente. Cette forme laisse donc voir l’image située à l’arrière-plan. Dans le programme, nous rendons la loupe mobile sur l’action de l’utilisateur. Mais cette loupe n’exerce rien, en tant que telle, sur l’image survolée. C’est, sous la loupe, que nous avons placé une image agrandie, dans un clip, clip sur lequel nous appliquons un masque avec ActionScript.

Il est à noter que les filtres n’affectent que les masques créés en ActionScript. C’est la raison pour laquelle nous procédons ainsi et non à partir des options de masque disponibles dans le

scénario. Dans le programme, plus loin, une action détermine le positionnement de ce clip en fonction du positionnement de la loupe. Pour affiner l'effet, nous ajoutons aussi au masque un filtre flou. C'est ce flou qui va permettre de simuler la distorsion de l'image à travers la lentille.

Pour réaliser ce dispositif, le code importe tout d'abord la sous-classe `BlurFilter` (flou) utilisée pour adoucir les bords du masque :

```
//----- initialisation
import flash.filters.BlurFilter;
```

Puis, nous définissons les premières instructions :

```
loupe_mc.cacheAsBitmap=true;
var flou:BlurFilter=new BlurFilter(10,10,3);
masque_mc.filters=[flou];
zoom_mc.mask=masque_mc;
var zoneDeplacement:Rectangle=new Rectangle(150,100,470,350);
```

Nous commençons par appliquer un lissage sur les formes vectorielles contenues dans la loupe à l'aide de la propriété `cacheAsBitmap` abordée dans la section précédente.

Nous appliquons ensuite un filtre flou sur le symbole `masque_mc`, utilisé pour révéler une partie de l'image agrandie. En appliquant une valeur de flou de quelques pixels seulement, nous définissons un contour assez fin pour le masque, et donc, une forme de lentille plutôt plate. En spécifiant des valeurs plus élevées, nous créons une forme de lentille plus convexe.

À la suite, nous attachons le symbole `masque_mc` à l'image agrandie, à l'aide de la propriété `mask` (`zoom_mc.mask=masque_mc`).

Nous définissons enfin un rectangle pour canaliser la zone de déplacement de la loupe. Nous limitons le déplacement à une surface proche de la surface de l'image, ceci afin d'éviter que l'utilisateur ne perçoive encore, dans les extrémités, l'image de dessous, ce qui serait incohérent d'un point de vue graphique.

Ensuite, viennent les actions qui définissent le déplacement de la loupe :

```
//----- actions
// déplacement de la loupe
loupe_mc.deplacerLoupe_btn.addEventListener(MouseEvent.CLICK, deplacerLoupe );
function deplacerLoupe(evt:MouseEvent) {
    loupe_mc.startDrag(false, zoneDeplacement);
}
addEventListener(MouseEvent.CLICK, arreterDeplacement );
function arreterDeplacement(evt:MouseEvent) {
    stopDrag();
}
```

La première fonction active le déplacement de la loupe dès que l'utilisateur appuie sur le bouton situé en haut et à droite de l'objet (`deplacementLoupe_btn`). Ce déplacement est défini par les paramètres `false` et `zoneDeplacement`. `false` désigne le fait que l'objet ne s'accroche pas sur le pointeur en son centre. Le déplacement est donc normal. La variable `zoneDeplacement` fait référence à la zone rectangulaire définie comme limites de la zone de déplacement.

La deuxième fonction interrompt tous les déplacements, quels qu'ils soient, dès que l'utilisateur relâche le bouton de la souris. Du fait que l'écouteur est placé sur la timeline du

scénario principal, l’effet de relâchement de souris est perçu aussi bien sur l’objet cliqué qu’en dehors.

La troisième partie affiche les commandes relatives au déplacement de l’image agrandie :

```
// positionnement de l'image zoomée
addEventListener(Event.ENTER_FRAME,positionMasque);
function positionMasque(evt:Event) {
    masque_mc.x=loupe_mc.x;
    masque_mc.y=loupe_mc.y;
    zoom_mc.x=( ( loupe_mc.x*-1)+(stage.stageWidth/2)) + (zoom_mc.width/2) )
    ➡ -loupe_mc.width;
    zoom_mc.y=( ( loupe_mc.y*-1)+(stage.stageHeight/2)) + (zoom_mc.height/2) )
    ➡ -loupe_mc.height;
}
```

Dans cette fonction exécutée continuellement (ENTER_FRAME), nous redéfinissons les positions des symboles masque_mc et zoom_mc.

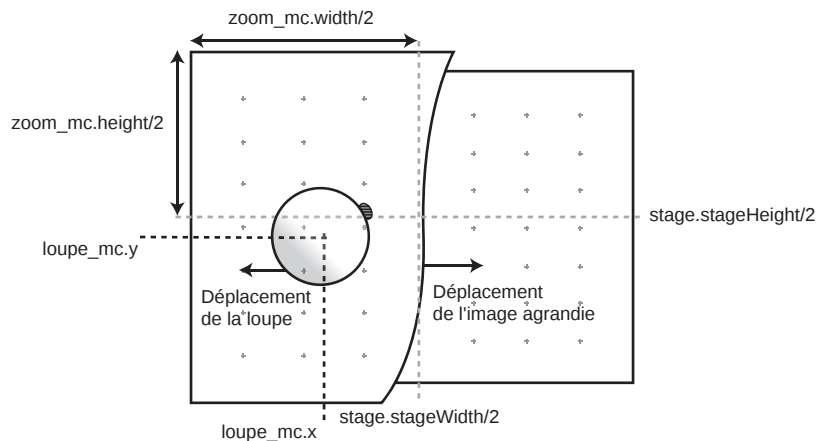
Le masque, affecté à l’image agrandie (zoom_mc), se cale sur la position de la loupe en X et Y. La zone visible de l’image agrandie suit donc le mouvement de la loupe. Les deux objets se superposent parfaitement, car leurs centres respectifs sont tous deux situés au milieu de la forme que chacun d’eux véhicule :

```
masque_mc.x=loupe_mc.x;
masque_mc.y=loupe_mc.y;
```

L’image agrandie (zoom_mc) est, quant à elle, positionnée en fonction de la loupe. L’équation que nous utilisons indique sommairement l’action suivante : la position de l’image agrandie (zoom_mc) correspond à la position de la loupe en valeur négative (voir Figure 11.9), plus la moitié de la scène (pour permettre d’inverser le coefficient lorsque le pointeur passe le centre du document), plus la demie largeur ou hauteur du zoom (pour recentrer l’image agrandie) moins les dimensions de la loupe (pour centrer l’image avec la loupe).

Figure 11.9

Mécanisme du positionnement de l’image agrandie.



Plus clairement, à travers cette équation étendue, nous désignons les actions suivantes :

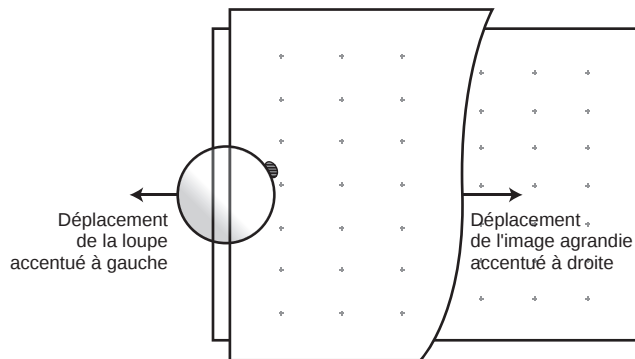
Lorsque nous déplaçons la loupe, si nous voulons que les extrémités des deux images coïncident pendant le déplacement, et que l'image agrandie ne se retrouve pas complètement décalée lorsque nous survolons les bords de l'image du dessous, nous devons définir sa position selon un coefficient. Ce coefficient élève ici le pas du déplacement de l'image agrandie en fonction de sa distance qui la sépare du centre du document. Lorsque la loupe part à gauche, nous élevons ainsi la valeur négative pour accélérer le pas du déplacement vers la gauche. Inversement, lorsque le déplacement s'effectue à droite, nous incrémentons la position de l'image agrandie en multipliant son pas, par un coefficient, cette fois-ci, positif.

Pour définir la valeur de ce coefficient, nous devons d'abord intégrer, dans notre calcul, les coordonnées du centre du document. C'est pourquoi nous utilisons `stage.stageWidth/2`, qui désigne le centre du document. Mais, nous employons aussi une valeur négative (`loupe_mc.x*-1`) pour que l'image agrandie se déplace en sens inverse de la loupe. Cela permet de faire coïncider les bords des deux images, lorsque la loupe atteint les extrémités de l'image originale (voir Figure 11.10) :

$$(\text{loupe_mc.x}*-1) + (\text{stage.stageWidth}/2)$$

Figure 11.10

Mécanisme du positionnement des images sur les bords.



Puis, nous ajoutons la moitié de la largeur de l'image agrandie, pour recentrer l'image dans la scène (`zoom_mc.width/2`). Nous retranchons, enfin, la moitié de la largeur de la loupe, pour recentrer le contenu par rapport à la loupe elle-même (`-loupe_mc.width`).

Nous déclinons ensuite le procédé pour la hauteur avec les propriétés `Y` et `height`.

Dans le fichier "ch11_apiGraphisme_3b.fla", nous proposons une variante de l'outil loupe. Dans ce document, la loupe suit directement le pointeur mais avec l'effet d'amortissement que nous avons étudié au Chapitre 1. Le code est le suivant :

```
addEventListener(Event.ENTER_FRAME,bougerLaBoule);

function bougerLaBoule (evt:Event) {
    boule_mc.x+=(mouseX-boule_mc.x)/20;
    boule_mc.y+=(mouseY-boule_mc.y)/20;
    ombre_mc.x=boule_mc.x;
    ombre_mc.y=boule_mc.y;
}
```

Pour en savoir plus sur les techniques d’animation en ActionScript, reportez-vous au Chapitre 1.

À retenir

- Pour créer une loupe, nous devons créer un masque flou, dynamiquement, afin de reconstituer la déformation de la lentille.
- Nous devons placer l’image dupliquée et masquée en fonction de la position de la loupe en faisant en sorte que les extrémités des deux images restent proches.
- Nous devons contraindre le mouvement de la loupe pour réserver l’affichage au contenu utile uniquement.
- Nous devons mettre en cache la forme graphique translucide pour optimiser les ressources de l’utilisateur.

Filtres de correction colorimétrique

Le moteur colorimétrique de Flash, affecté à la classe `colorMatrix`, permet de modifier la valeur de chaque pixel qui compose l’image. Il est donc possible d’appliquer des filtres pour modifier l’aspect d’une image ou en extraire certaines couches, en vue d’une exploitation pour le relief, par exemple. Dans le contexte de développements plus simples, la classe `colorMatrix` permet aussi d’appliquer des filtres de saturation, de désaturation, de luminosité, sur les images. Elle permet de décliner dans une même interface différentes occurrences d’une image et de créer, à moindre poids, des décors composites complexes et aléatoires (une ville avec des maisons de nuances différentes, une forêt avec des arbres différents, un trafic routier avec des véhicules de teintes différentes, mais composés à partir des mêmes symboles).

Dans cette section, nous allons voir comment appliquer et restaurer un filtre sur une image. Sur le même principe, nous abordons aussi la manière d’appliquer dynamiquement un filtre à plusieurs occurrences d’une même composition, mais avec des valeurs différentes. Nous voyons enfin comment animer un filtre de sorte qu’il apparaisse progressivement, grâce à ActionScript.

Appliquer et restaurer un filtre



Exemples > `ch11_apiGraphisme_4 fla`

Dans le document "`ch11_apiGraphisme_4 fla`", sur la scène principale, apparaît un Movie-Clip qui contient une image. À droite, figure un bouton de restauration (voir Figure 11.11).

Figure 11.11

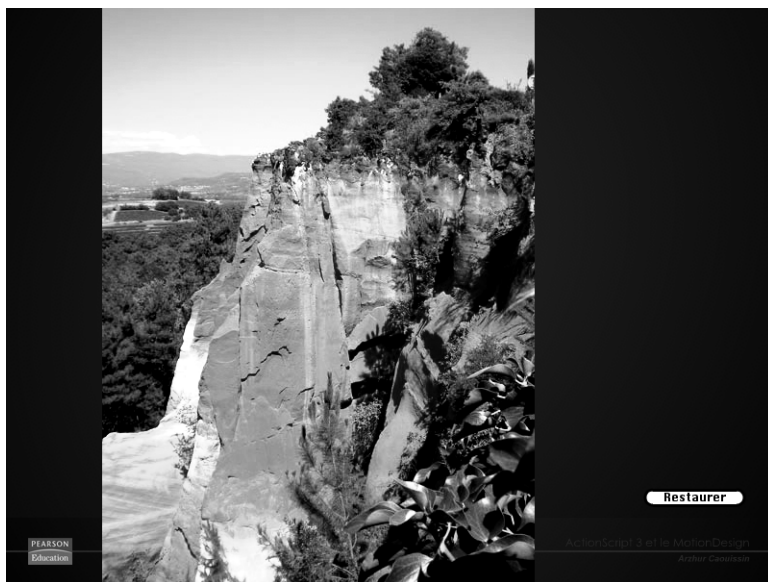
Aperçu de la scène principale, image normale.



En publiant le document, l'image est aussitôt modifiée. Un filtre de saturation est automatiquement appliqué. Lorsque le bouton Restaurer est cliqué, l'image revient à l'état initial (voir Figure 11.12).

Figure 11.12

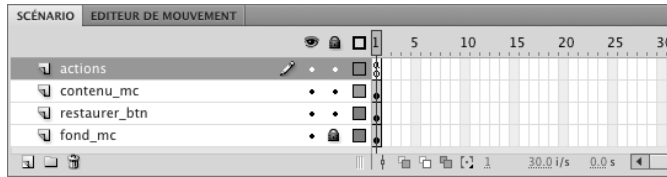
Aperçu du document publié, image saturée.



Dans le scénario de la scène principale, au-dessus du calque `fond_mc`, nous distinguons les calques `contenu_mc` et `restaurer_btn`, qui correspondent aux deux objets placés sur la scène (voir Figure 11.13).

Figure 11.13

Aperçu du
scénario de la
scène principale.



Le calque actions affiche le code suivant :

```
//----- initialisation
import flash.filters.ColorMatrixFilter;

//----- actions
// filtre saturation
var coef1:Number=2;
var coef255:int=-100;
var tableau1:Array = new Array();
tableau1 = new Array();
tableau1=tableau1.concat([coef1,0,0,0,coef255]);
tableau1=tableau1.concat([0,coef1,0,0,coef255]);
tableau1=tableau1.concat([0,0,coef1,0,coef255]);
tableau1=tableau1.concat([0,0,0,1,0]);
//
var filtreSaturation:ColorMatrixFilter=new ColorMatrixFilter(tableau1);
contenu_mc.filters=[filtreSaturation];

// filtre restauration
var tableau2:Array = new Array();
tableau2=tableau2.concat([1,0,0,0,0]);
tableau2=tableau2.concat([0,1,0,0,0]);
tableau2=tableau2.concat([0,0,1,0,0]);
tableau2=tableau2.concat([0,0,0,1,0]);
var filtreNegatif:ColorMatrixFilter=new ColorMatrixFilter(tableau2);
//
restaurer_btn.addEventListener(MouseEvent.CLICK,appliquerFiltreNegatif);
function appliquerFiltreNegatif(evt:MouseEvent) {
    contenu_mc.filters=[filtreNegatif];
}
```

Dans ce code, nous appliquons un filtre, dynamiquement. Nous l’appliquons à nouveau, sur action de l’utilisateur, en modifiant simplement les valeurs de sorte que l’image affectée soit restaurée.

Les filtres colorimétriques appartiennent à la classe `filters` et plus spécifiquement à la sous-classe `ColorMatrixFilter`. Nous l’importons via l’instruction suivante :

```
//----- initialisation
import flash.filters.ColorMatrixFilter;
```

Plus bas, nous appliquons un filtre dont les valeurs définissent une augmentation de la saturation :

```
//----- actions
// filtre saturation
var coef1:Number=2;
```

```

var coef255:int=-100;
var tableau1:Array = new Array();
tableau1 = new Array();
tableau1=tableau1.concat([coef1,0,0,0,coef255]);
tableau1=tableau1.concat([0,coef1,0,0,coef255]);
tableau1=tableau1.concat([0,0,coef1,0,coef255]);
tableau1=tableau1.concat([0,0,0,1,0]);
//
var filtreSaturation:ColorMatrixFilter=new ColorMatrixFilter(tableau1);
contenu_mc.filters=[filtreSaturation];

```

Un filtre colorimétrique, en ActionScript, se définit par une matrice de valeurs appliquée à chaque pixel de l'image pour laquelle il est affecté. Cette matrice est composée d'une série de cinq valeurs (Rouge, Vert, Bleu, Alpha et Luminosité) définissant le décalage de positionnement de chaque pixel, dans le cercle chromatique, et ce pour chacune des quatre teintes (Rouge, Vert, Bleu et Alpha). Ainsi, nous obtenons un tableau composé de vingt valeurs (5×4).

Le tableau de valeurs, utilisé par la matrice de couleurs, peut également adopter la forme suivante :

```
nomDuTableau = [1,0,0,0,255,0,1,0,0,255,0,0,1,0,255,0,0,0,1,0].
```

Pour des raisons de clarté, nous préférons employer la structure éclatée, qui revient strictement au même résultat. Cette structure offre juste l'avantage de permettre une identification plus rapide des valeurs appliquées distinctement pour chaque teinte (R, V, B et A) :

```

var nomDuTableau : Array = new Array();
nomDuTableau = new Array();
nomDuTableau =nomDuTableau.concat([1,0,0,0,255]); //R
nomDuTableau =nomDuTableau.concat([0,1,0,0,255]); //V
nomDuTableau =nomDuTableau.concat([0,0,1,0,255]); //B
nomDuTableau =nomDuTableau.concat([0,0,0,1,0]); //A

```

Chaque ligne utilise la méthode `concat()` qui ajoute de nouvelles valeurs au tableau. Nous disons que les valeurs sont alors concaténées aux valeurs précédentes.

Les valeurs utilisées pour définir l'intensité du rouge, du vert et du bleu de chaque teinte RVBA sont définies par un pourcentage. Ce pourcentage est de type décimal. Pour indiquer une intensité normale, nous inscrivons donc la valeur 1, qui correspond à 100 %. Pour désigner une valeur nulle, nous inscrivons 0, qui correspond à 0 %. Toutes les valeurs décimales intermédiaires permettent de définir des nuances de couleurs intermédiaires. Les valeurs peuvent être supérieures à 1 et, même, négatives. Dans le cas de valeurs négatives, la teinte modifiée bascule, dans le cercle chromatique, vers une nuance complémentaire. Étant donné que le calcul multiplie les valeurs entre elles, une modification *a priori* bénigne peut souvent aboutir à des surprises et notamment à l'assombrissement intégral de l'image. Travaillez donc en mesurant bien ces valeurs.

Les valeurs de luminosité, ajoutées en fin de ligne, sont définies entre -255 et +255. Une valeur de 0 ne modifie pas la luminosité de l'image. Une valeur qui évolue vers -255 ajoute du noir à l'image. Une valeur qui évolue vers +255 ajoute du blanc.

Ainsi, nous pouvons définir plusieurs types de réglages à partir de la même structure. Si nous augmentons les valeurs de luminosité tout en passant les valeurs de teinte en négatif, nous saturons l'image.

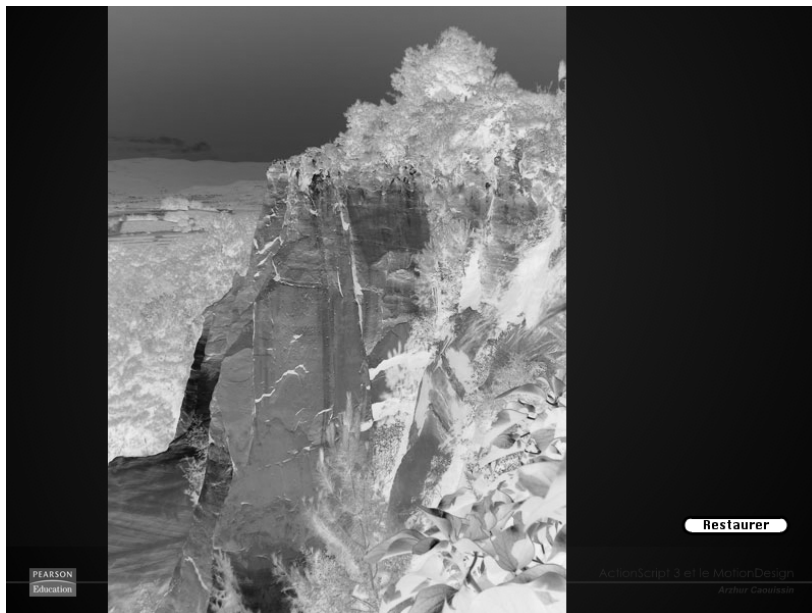
Vous trouverez, ci-après, différentes variations du filtre colorimétrique en fonction du type de réglage voulu.

Filtre négatif (voir Figure 11.14) :

- `tableau1=tableau1.concat([-1,0,0,0,255]);`
- `tableau1=tableau1.concat([0,-1,0,0,255]);`
- `tableau1=tableau1.concat([0,0,-1,0,255]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

Figure 11.14

Aperçu du filtre négatif.



Filtre Rouge :

- `tableau1=tableau1.concat([-1,0,0,0,0]);`
- `tableau1=tableau1.concat([0,-1,0,0,-255]);`
- `tableau1=tableau1.concat([0,0,-1,0,-255]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

Filtre Vert :

- `tableau1=tableau1.concat([-1,0,0,0,-255]);`
- `tableau1=tableau1.concat([0,-1,0,0,0]);`
- `tableau1=tableau1.concat([0,0,-1,0,-255]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

Filtre Bleu :

- `tableau1=tableau1.concat([-1,0,0,0,-255]);`
- `tableau1=tableau1.concat([0,-1,0,0,-255]);`
- `tableau1=tableau1.concat([0,0,-1,0,0]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

Pour désigner des filtres de couleurs complémentaires RVB (ou primaires CMJ), inscrivez seulement une valeur de luminosité négative par tableau, au lieu de deux pour les filtres primaires :

Filtre Cyan (complémentaire du Rouge en RVB) :

- `tableau1=tableau1.concat([-1,0,0,0,-255]);`
- `tableau1=tableau1.concat([0,-1,0,0,0]);`
- `tableau1=tableau1.concat([0,0,-1,0,0]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

Filtre Magenta (complémentaire du Vert en RVB) :

- `tableau1=tableau1.concat([-1,0,0,0,0]);`
- `tableau1=tableau1.concat([0,-1,0,0,-255]);`
- `tableau1=tableau1.concat([0,0,-1,0,0]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

Filtre Jaune (complémentaire du Bleu en RVB) :

- `tableau1=tableau1.concat([-1,0,0,0,0]);`
- `tableau1=tableau1.concat([0,-1,0,0,0]);`
- `tableau1=tableau1.concat([0,0,-1,0,-255]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`

En appliquant des valeurs sur les autres paramètres de couleur, à la place des 0 affichés habituellement, nous décalons radicalement les couleurs de l'image dans le cercle chromatique tout en préservant cependant leur luminosité :

Filtre mélangeur de couleurs :

- `tableau1=tableau1.concat([1,1,1,0,255]);`
- `tableau1=tableau1.concat([0,1,0,0,255]);`
- `tableau1=tableau1.concat([0,0,1,0,255]);`
- `tableau1=tableau1.concat([0,0,0,1,0]);`



Dans notre exemple, le premier tableau applique une saturation en doublant les valeurs de teinte (où coef1 vaut 2, soit 200 % de saturation). Pour compenser l’assombrissement obtenu de l’image, nous réduisons également sa luminosité (où coef255 vaut 100) :

```
// filtre saturation
var coef1:Number=2;
var coef255:int=-100;
var tableau1:Array = new Array();
tableau1 = new Array();
tableau1=tableau1.concat([coef1,0,0,0,coef255]);
tableau1=tableau1.concat([0,coef1,0,0,coef255]);
tableau1=tableau1.concat([0,0,coef1,0,coef255]);
tableau1=tableau1.concat([0,0,0,1,0]);
//
var filtreSaturation:ColorMatrixFilter=new ColorMatrixFilter(tableau1);
contenu_mc.filters=[filtreSaturation];
```

Le tableau est ensuite introduit en paramètre de la méthode ColorMatrixFilter pour en faire un filtre colorimétrique. Pour appliquer le filtre colorimétrique à une sélection, nous utilisons la propriété filters qui désigne la matrice en question.

Le principe est le même pour restaurer les valeurs colorimétriques initiales. Nous isolons simplement l’action du filtre dans une fonction, associée à un événement souris. Les paramètres du tableau reprennent alors des valeurs neutres :

```
// filtre restauration
var tableau2:Array = new Array();
tableau2=tableau2.concat([1,0,0,0,0]);
tableau2=tableau2.concat([0,1,0,0,0]);
tableau2=tableau2.concat([0,0,1,0,0]);
tableau2=tableau2.concat([0,0,0,1,0]);
var filtreNegatif:ColorMatrixFilter=new ColorMatrixFilter(tableau2);
//
restaurer_btn.addEventListener(MouseEvent.CLICK,appliquerFiltreNegatif);
function appliquerFiltreNegatif(evt:MouseEvent) {
    contenu_mc.filters=[filtreNegatif];
}
```



Résolution limite des images. Flash peut appliquer des filtres et traiter des images de résolution maximum de 16 777 216 pixels. Soit, pour une image de ratio 1 sur 4, faisant 8 192 pixels de large, Flash peut traiter une hauteur maximale de 2 048 pixels.

Correction colorimétrique par lot



Exemples > ch11_apiGraphisme_5 fla

Dans le document "ch11_apiGraphisme_5 fla", sur la scène principale, apparaît un Movie-Clip qui contient plusieurs occurrences d’un clip de forme graphique, importée d’Illustrator (voir Figure 11.15).

Figure 11.15

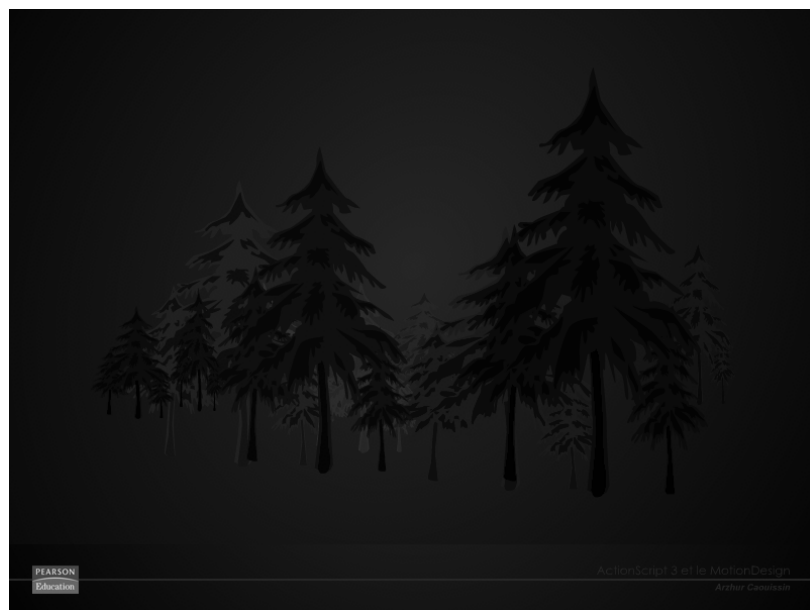
Aperçu de la scène principale, image normale.



En publiant le document, un filtre applique dynamiquement un effet différent pour chaque occurrence d'objet (voir Figure 11.16).

Figure 11.16

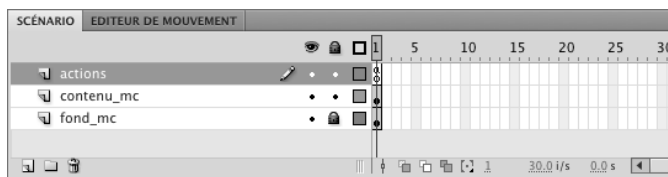
Aperçu du document publié, filtre appliqué par lot.



Dans le scénario, le calque contenu_mc accueille le MovieClip de toutes les occurrences d'arbres (voir Figure 11.17).

Figure 11.17

Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//----- initialisation
import flash.filters.ColorMatrixFilter;

//----- actions
//
var coef1:Number;
var coef255:int;
var tableau1:Array = new Array();
var filtreSaturation:ColorMatrixFilter=new ColorMatrixFilter(tableau1);
//
for (var i:Number=0; i<10; i++) {
    coef1=i*-0.15;
    coef255=i*2+50;
    tableau1 = new Array();
    tableau1=tableau1.concat([coef1,0,0,0,coef255]);
    tableau1=tableau1.concat([0,coef1,0,0,coef255]);
    tableau1=tableau1.concat([0,0,coef1,0,coef255]);
    tableau1=tableau1.concat([0,0,0,1,0]);
    filtreSaturation=new ColorMatrixFilter(tableau1);
    contenu_mc.getChildAt(i).filters=[filtreSaturation];
}
```

Ici, nous définissons les variables requises pour la génération du filtre. À la différence de la section précédente, nous isolons simplement le tableau et modifions dynamiquement ses valeurs, en intégrant les données dans une boucle for. Il nous suffit alors de récupérer la valeur d’itération (i) pour démultiplier les valeurs passées en paramètre du filtre. Nous utilisons également i pour appliquer, dynamiquement, chaque filtre à un objet du symbole contenu_mc, par ordre de la liste d’affichage (getChildAt(i)).

Correction colorimétrique par interpolation



Exemples > ch11_apiGraphisme_6.fla

Dans le document "ch11_apiGraphisme_6.fla", nous retrouvons la même structure que dans le document précédent. En le publiant, un filtre s’applique dynamiquement, mais il évolue ici de manière autonome et en fonction de la position du pointeur en X (voir Figure 11.18).

Figure 11.18

Aperçu du document publié.



Dans la fenêtre de scénario, le calque actions affiche le code suivant :

```
//----- initialisation
import flash.filters.ColorMatrixFilter;

//----- actions
//
var coef1:Number=-2;
var coef255:int=100;
var tableau1:Array = new Array();
var filtreSaturation:ColorMatrixFilter=new ColorMatrixFilter(tableau1);
//
addEventListener(Event.ENTER_FRAME,animerFiltre);
function animerFiltre (evt:Event) {
    if (coef1<2) {
        coef1=coef1+0.01;
    }
    coef255=100+(mouseX/30);
    for (var i:Number=0; i<10; i++) {
        tableau1 = new Array();
        tableau1=tableau1.concat([coef1,0,0,0,coef255]);
        tableau1=tableau1.concat([0,coef1,0,0,coef255]);
        tableau1=tableau1.concat([0,0,coef1,0,coef255]);
        tableau1=tableau1.concat([0,0,0,1,0]);
        filtreSaturation=new ColorMatrixFilter(tableau1);
        contenu_mc.getChildAt(i).filters=[filtreSaturation];
    }
}
```

Dans ce programme, nous plaçons le filtre à l'intérieur d'une fonction appelée par un gestionnaire de type ENTER_FRAME. Il nous est alors possible de modifier dynamiquement les valeurs passées en paramètre du tableau de la matrice de couleurs.

La première valeur, coef1, est modifiée par incrémentation. Une condition définit une limite cependant à l'incrémentation, de manière à ce que la valeur n'excède pas 2 :

```
if (coef1<2) {
    coef1=coef1+0.01;
}
```

La seconde valeur, `coef255`, est modifiée en fonction de la position du pointeur dans le document :

```
coef255=100+(mouseX/30);
```



Animation de filtres. Reportez-vous également au Chapitre 3 pour en savoir plus sur la gestion de filtres avec des interpolations de type TweenMax.

À retenir

- Nous pouvons appliquer des filtres colorimétriques pour modifier ponctuellement les images d’une composition. Pour cela, nous utilisons la classe `ColorMatrixFilter`.
- Il est possible d’appliquer un filtre par lot, sur plusieurs occurrences d’objets, dynamiquement. Pour cela, nous plaçons le filtre dans une boucle `for` et utilisons la valeur d’itération de `i` pour modifier les paramètres du filtre et définir les objets à modifier.
- Il est possible d’animer un filtre en le plaçant directement dans un gestionnaire de type `ENTER_FRAME`. Nous pouvons alors animer automatiquement le filtre par incrémentation de valeur, mais aussi en fonction d’autres paramètres comme la position du pointeur ou d’un objet (type curseur), par exemple.

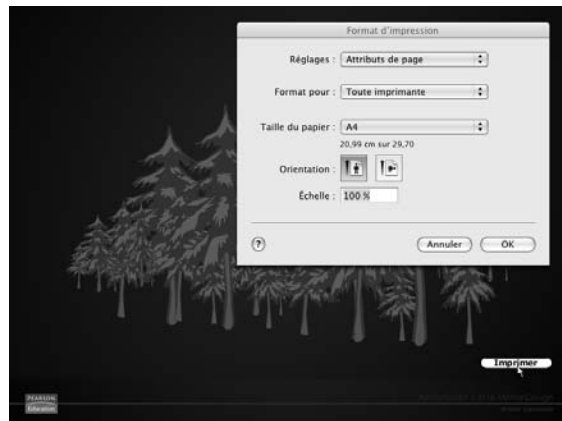
Imprimer un document SWF

Vous pouvez imprimer directement tout type de contenu dès lors que celui-ci possède un conteneur. Il suffit d’invoquer la méthode `PrintJob()` et de cibler le conteneur à imprimer pour lancer la fenêtre d’impression automatiquement.

Dans cette section, nous plaçons des instructions d’impression, sur le document précédent, de manière à imprimer l’image affichée (voir Figure 11.19).

Figure 11.19

Aperçu du document publié et exécuté pour l’impression.





Exemples > ch11_apiGraphisme_7 fla

Dans la scène principale du document "ch11_apiGraphisme_7 fla", nous distinguons le bouton d'impression et le conteneur contenu_mc. Dans la fenêtre de scénario, ils sont toujours bien répartis vers les calques. Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//imprimer
imprimer_btn.addEventListener(MouseEvent.CLICK,imprimer);
function imprimer(evt:MouseEvent) {
    var impression:PrintJob = new PrintJob();
    if (impression.start()) {
        if (contenu_mc.width>impression.pageWidth) {
            contenu_mc.width=impression.pageWidth;
            contenu_mc.scaleY=contenu_mc.scaleX;
        }
        impression.addPage(contenu_mc);
        impression.send();
    }
}
```

La classe d'impression emploie plusieurs méthodes. D'abord, nous ouvrons la boîte de dialogue d'impression (start) et, seulement si celle-ci est rendue disponible par le système (if (impression.start())), alors, nous indiquons les pages à imprimer.

Dans le bloc d'instructions, nous précisons les dimensions d'impression en hauteur et en largeur. Ces dimensions sont définies en pixels, bien que, en règle générale, l'impression se mesure, elle, en points. Les dimensions de la zone imprimable sont détectées automatiquement et ne peuvent être modifiées. Pour connaître les dimensions ces valeurs, nous utilisons les propriétés pageWidth et pageHeight, en lecture seule, sur l'objet PrintJob(impression.pageWidth).

En l'occurrence, nous spécifions que si la largeur du contenu à imprimer est supérieure à celle de la zone d'impression, alors, nous l'adaptions à la zone d'impression (contenu_mc.width=impression.pageWidth). Afin de conserver toutefois l'homothétie du contenu à imprimer, nous ajoutons une instruction qui adapte également l'échelle de la hauteur du contenu en fonction de sa nouvelle largeur (contenu_mc.scaleX=contenu_mc.scaleY).

Puis, à l'aide de la méthode addpage(), nous spécifions le conteneur à imprimer. Cette méthode peut être multipliée autant de fois que de conteneurs à imprimer.

Le programme termine la gestion de l'impression en envoyant le tout dans la file d'attente de l'imprimante, avec la méthode send().

La classe printJob permet d'imprimer des conteneurs, mais aussi des zones bitmap telles que nous les avons définies dans ce chapitre. Il suffit alors de remplacer le nom du conteneur par l'objet Bitmap à imprimer.



Options d'impression. Pour le détail des commandes liées à l'impression de documents SWF (mise à l'échelle, gestion des pages, affichage horizontal ou vertical), consultez aussi l'aide en ligne, pour une fois, limpide à ce sujet : http://livedocs.adobe.com/flash/9.0_fr/ActionScriptLangRefV3/flash/printing/PrintJob.html.

À retenir

- Il est possible d’imprimer un contenu Flash, même isolé dans un symbole, y compris si celui-ci a été modifié en ActionScript. Nous utilisons pour cela la classe `printJob()`.
- Lorsqu’un contenu est de dimension supérieure à la zone d’impression, nous le redimensionnons à l’échelle de la zone d’impression à l’aide des propriétés `pageWidth` ou `pageHeight`.

Appliquer une teinte aléatoire

Dans de nombreuses circonstances, un contenu doit pouvoir changer de teinte, de manière totalement opaque : convertir un bouton en lien visité, modifier la teinte d’une région survolée dans une carte géographique interactive, mettre à jour la couleur de l’ensemble des textes d’un site lorsqu’un nouveau thème artistique l’impose.

Dans cette section, nous appliquons une couleur à notre conteneur central en cliquant sur un simple bouton. Nous spécifions, en outre, une valeur aléatoire afin de désigner une couleur différente à chaque clic (voir Figure 11.20).

Figure 11.20

Aperçu du document publié.



Exemples > ch11_apiGraphisme_8 fla

Le programme affiché dans la fenêtre Actions est le suivant :

```
//teinter
teinte_btn.addEventListener(MouseEvent.CLICK,teinter);
function teinter (evt:MouseEvent) {
```

```
var teinte:ColorTransform = new ColorTransform ();  
teinte.color=0xffffffff*Math.random();  
contenu_mc.transform.colorTransform=teinte;  
}
```

En cliquant sur le bouton `teinte_btn`, nous créons une nouvelle variable `teinte` qui matérialise un objet `ColorTransform`. C'est cet objet qui permet de véhiculer la teinte pour l'appliquer ensuite à tout type de contenu.

Pour cet objet `ColorTransform`, nous modifions ensuite la propriété `color`, qui désigne une couleur hexadécimale.

Nous pourrions directement définir une couleur hexadécimale, mais nous la multiplions ici par une valeur aléatoire, générée par la classe `Math.random()`, vue précédemment.

Nous devons comprendre, au sujet de la valeur aléatoire, qu'une couleur hexadécimale est en fait une simple valeur numérique même si elle semble composée de caractères alphanumériques (voir note "Système hexadécimal"). En la multipliant par une autre valeur numérique, nous modifions par conséquent la teinte affichée. Notons également que la valeur utilisée initialement est le blanc (`ffffff`), car le blanc désigne la valeur numérique maximale pour une teinte (255,255,255). En la multipliant par une valeur décimale aléatoire, comprise entre 0 et 1, nous obtenons toutes les nuances de couleur comprises entre 0,0,0 (noir) et 255,255,255 (blanc). La valeur `0x` indique, elle, au compilateur qu'il s'agit d'une valeur hexadécimale. C'est précisément ce qui lui permet de convertir une suite de lettres et de chiffres en valeur numérique.



Système hexadécimal. Les nombres sont généralement codés en décimal, c'est-à-dire en base 10. Les couleurs web sont, elles, codées en hexadécimal, soit sur une base 16 (ou 2 élevé deux fois au carré). Chaque valeur hexadécimale est donc une valeur numérique définie sur 16 bits. Les six valeurs qui dépassent la dizaine sont simplement transcrites sous la forme de caractères texte, mais leur valeur est bien numérique.

Pour terminer le programme, nous appliquons la teinte à l'objet en utilisant la méthode `colorTransform()` de la propriété `transform`. C'est la méthode `colorTransform()` qui copie les pixels créés. La propriété `transform` les applique à l'objet `contenu_mc`.

La classe `ColorTransform` peut aussi affecter une zone partielle de l'objet. Pour en savoir plus sur cette option clairement exposée ici, consultez aussi l'aide en ligne à l'adresse suivante : [http://help.adobe.com/fr_FR/AS3LCR/Flash_10.0/flash/display/BitmapData.html#colorTransform\(\)](http://help.adobe.com/fr_FR/AS3LCR/Flash_10.0/flash/display/BitmapData.html#colorTransform()).

À retenir

- Pour modifier intégralement la teinte d'un objet, nous utilisons la classe `colorTransform`.
- Pour modifier dynamiquement une valeur hexadécimale, il suffit de la multiplier par un coefficient numérique.

Créer un puits de couleurs

L’affectation d’une teinte à un objet peut s’avérer utile si l’on permet à l’utilisateur de prélever lui-même cette teinte sur un nuancier ou sur une image de son choix. Dans cet exemple, nous plaçons un nuancier sur la scène principale et nous permettons à l’utilisateur d’y sélectionner une couleur. Aussitôt la couleur activée, l’illustration l’affiche (voir Figure 11.21).

Figure 11.21

Aperçu du document publié.



Exemples > ch11_apiGraphisme_9 fla

Dans ce document, la scène affiche un symbole contenu_mc qui contient une série de Movie-Clip. À droite, hors champ, apparaît un autre symbole nommé couleurs_mc. À l’intérieur, au dernier niveau d’imbrication, trois objets sont distinctement répartis vers les calques. À l’arrière-plan, figure un rectangle transparent (de valeur alpha 0). Au-dessus, sur un troisième calque, prend place la forme sur laquelle repose un nuancier de couleurs (voir Figure 11.22).

Figure 11.22

Nuancier de couleurs.



Sur la scène principale, dans le calque actions (voir Figure 11.23), nous pouvons lire :

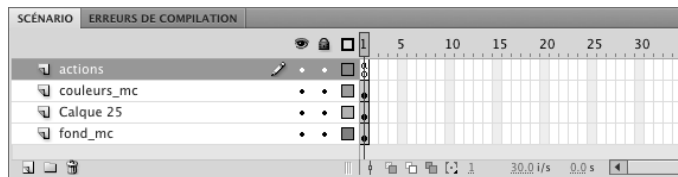
```
// affichage de la palette
var monImage:BitmapData=new
BitmapData(stage.stageWidth,stage.stageHeight,true,0x00000000);
monImage.draw(couleurs_mc.getChildAt(0));
var imageDepart:Bitmap=new Bitmap(monImage);
var capture:BitmapData=imageDepart.bitmapData;
var miroir:Bitmap=new Bitmap(capture);
addChild(miroir);

//zone cliquable
var initX:Number=couleurs_mc.zone_mc.nuancier_mc.x;
var initY:Number=couleurs_mc.zone_mc.nuancier_mc.y;
var initWidth:Number=couleurs_mc.zone_mc.nuancier_mc.width;
var initHeight:Number=couleurs_mc.zone_mc.nuancier_mc.height;

// application de la couleur choisie
addEventListener(MouseEvent.CLICK,pipette);
function pipette(evt:MouseEvent) {
    if (mouseX>initX && mouseY>initY && mouseX<initX+initWidth && mouseY
    <initY+initHeight) {
        // capturer
        var couleurDuPixel:uint=capture.getPixel(mouseX,mouseY);
        var couleur:*= "0x"+couleurDuPixel.toString(16);
        // teinter
        var teinte:ColorTransform = new ColorTransform ();
        teinte.color=couleur;
        contenu_mc.transform.colorTransform=teinte;
    }
}
```

Figure 11.23

Scénario de la scène principale.



Pour comprendre le mécanisme d'un puits de couleurs, nous devons savoir que la méthode `getPixel` utilisée pour prélever une couleur sur un objet, renvoie uniquement la couleur d'un objet `BitmapData`. Nous devons par conséquent afficher cet objet avant de pouvoir y récupérer un pixel de couleur. C'est seulement ensuite que nous redistribuons le prélèvement effectué sur un autre objet, grâce à un filtre de couleur.

Dans cet exemple, nous utilisons le pointeur de la souris, pour cibler la couleur à prélever.

Les premières lignes de code activent l'objet `BitmapData` qui copie le contenu graphique du symbole `couleurs_mc` situé hors champs. L'image capturée est réaffectée à un nouvel objet `Bitmap` ajouté à la liste d'affichage, au point d'origine du document :

```
// affichage de la palette
var monImage:BitmapData=new
BitmapData(stage.stageWidth,stage.stageHeight,true,0x00000000);
monImage.draw(couleurs_mc.getChildAt(0));
var imageDepart:Bitmap=new Bitmap(monImage);
var capture:BitmapData=imageDepart.bitmapData;
var miroir:Bitmap=new Bitmap(capture);
addChild(miroir);
```

Il est important de souligner que la zone d’affectation du BitmapData est définie par les deux premières valeurs de la méthode BitmapData. Ainsi, pour réduire ou modifier la zone d’affichage du puits de couleurs, vous devez modifier également ces valeurs.

Dans la suite du code, nous définissons la zone d’action pour le prélèvement de la couleur. Cette zone reprend les coordonnées exactes et les dimensions du nuancier :

```
//zone cliquable
var initX:Number=couleurs_mc.zone_mc.nuancier_mc.x;
var initY:Number=couleurs_mc.zone_mc.nuancier_mc.y;
var initWidth:Number=couleurs_mc.zone_mc.nuancier_mc.width;
var initHeight:Number=couleurs_mc.zone_mc.nuancier_mc.height;
```

Enfin, apparaît la fonction pipette qui exécute le prélèvement et l’affectation de la couleur :

```
// application de la couleur choisie
addEventListener(MouseEvent.CLICK,pipette);
function pipette(evt:MouseEvent) {
    if (mouseX>initX && mouseY>initY && mouseX<initX+initWidth && mouseY
    <initY+initHeight) {
        // capturer
        var couleurDuPixel:uint=capture.getPixel(mouseX,mouseY);
        var couleur:*= "0x"+couleurDuPixel.toString(16);
        // teinter
        var teinte:ColorTransform = new ColorTransform ();
        teinte.color=couleur;
        contenu_mc.transform.colorTransform=teinte;
    }
}
```

Dans la fonction pipette, nous commençons par capturer la teinte sur la zone définie par l’objet BitmapData, et donc, par le nuancier, grâce à la méthode getPixel(). En paramètre de cette méthode, nous spécifions les coordonnées du pointeur au moment du clic (mouseX et mouseY).

La méthode getPixel() prélève une couleur définie en RVB. Pour affecter cette valeur à un filtre colorimétrique, nous devons la convertir d’abord en valeur hexadécimale.

La deuxième instruction enregistre donc cette valeur RVB en la précédant des caractères zéro et x, qui désignent, en ActionScript, une valeur hexadécimale. Afin de convertir la valeur RVB en valeur également hexadécimale (en base de 16), nous utilisons le transtypage toString(16).

La seconde partie de cette fonction emploie la méthode `ColorTransform` que nous connaissons déjà, et applique la teinte à l'objet `contenu_mc`, présent sur la scène.

À retenir

- Pour prélever une couleur, nous utilisons la méthode `getPixel()` qui fonctionne avec un objet `BitmapData`.
- Pour que la couleur prélevée corresponde à l'endroit cliqué, nous passons, en paramètre de la méthode `getPixel`, les coordonnées X et Y du pointeur.
- Pour affecter enfin la teinte à un objet, nous appliquons le filtre colorimétrique `ColorTransform`.

Synthèse

Dans ce chapitre, vous avez appris à modifier intrinsèquement le contenu graphique des images et des formes vectorielles. Vous avez appris à optimiser la gestion de contenus graphiques en déclinant certains objets par simple modification de couleur. Vous avez appris à recréer des effets d'optique déformante et à lisser les images affichées, même lorsqu'elles sont interpolées. Vous avez enfin appris à réaliser des animations de filtres en vue la création de systèmes graphiques élaborés.

Vous savez désormais traiter les images de façon à les valoriser et vous disposez d'outils déclinables dans des interpolations animées (galeries d'images) ou pour des mises en forme plus spécifiques, comme celle abordée au prochain chapitre.

Introduction

L'impression de relief est créée par le cerveau. Il analyse les deux images reçues par chaque œil et reconstitue le volume en superposant ces deux images. Pour créer des images en relief, nous devons donc organiser l'affichage de sorte que le cerveau puisse lire deux sources d'images distinctes, dans un temps donné, le plus court possible.

Il existe différentes techniques de montage en relief. La plus accessible, tous supports confondus, demeure la technique de l'anaglyphe. L'anaglyphe se caractérise par la création d'une image composée de couches séparées : d'une part, le rouge et d'autre part, le vert et bleu. Le rouge est dissocié du vert et du bleu qui eux, restent superposés. Cette technique implique certes l'utilisation de lunettes rouges et bleues (ou rouges et vertes) pour l'interpréter, mais peut se déployer aussi bien sur un écran vidéo, informatique, cinéma, que sur le papier (exemples d'anaglyphes : http://www.dailymotion.com/video/xb6ez7_visitez-paris-en-3d-16_creation).

Les autres techniques demeurent plus complexes et sont généralement réservées à des dispositifs plus lourds à mettre en œuvre, mais qui répondent à une exigence manifeste sur le choix et la qualité des couleurs. Ainsi, sont également rencontrées :

- **La technique des lunettes passives.** Elle consiste à employer une paire de lunettes dont chaque verre affiche une trame soit horizontale soit verticale. L'image projetée est dissociée à travers l'utilisation de deux projecteurs. Chacun affiche une image pour une trame donnée, en reproduisant la trame correspondant à l'œil visé.
- **La technique des lunettes actives.** Elle consiste à placer un écran translucide à cristaux liquides devant chaque œil. Contrôlé par un système de laser qui les relie à l'écran, chaque verre alterne le passage de la lumière pour ne laisser passer que l'image qui correspond à l'œil visé. L'image est synchrone avec le rythme de balayage de la lunette. Ces lunettes requièrent une légère alimentation électrique généralement assumée par l'utilisation de piles. Cette technique est favorisée dans la production audiovisuelle car elle ne limite pas le choix des couleurs lors de la capture, contrairement à l'anaglyphe pour lequel ce qui est trop rouge ou trop bleu, du fait du principe de couches séparées, ne peut être traité en relief.
- **La technique du réseau lenticulaire.** Cette technique ne requiert pas de lunettes. Il s'agit de placer sur un écran une couche uniforme de lentilles convexes très fines, collées les unes à la suite des autres. L'image affichée est décomposée en fines trames à raison de 8 à 12 trames différentes par lentille. Il faut donc capturer 8 à 12 images pour reconstituer un volume. C'est en se déplaçant le long de l'écran lenticulaire que le volume seulement réapparaît, et à une distance bien définie. Ce dispositif complexe n'est utilisé que pour les images fixes dans des zones d'exposition à faible recul et pour des usagers en mouvement (vitrines, galeries ou salons).

Dans ce chapitre, nous abordons la création d'images en relief de type anaglyphe, avec Photoshop, puis, la gestion de l'affichage de ces images en ActionScript. Nous présentons, en fin de chapitre, un moteur de rendu en relief, entièrement dynamique, qui permet de convertir automatiquement en véritable objet en relief tout objet placé dans un espace 3D dans Flash.

Pour étudier les exemples de ce chapitre, il est recommandé de disposer de lunettes pour anaglyphes, à films rouge et bleu, disponibles dans votre kiosque, chez votre marchand de jouet ou sur Internet. Certains opticiens et photographes proposent également ce type d'équipement (voir aussi <http://www.alpes-stereo.com/lunettes.html>).

Prise de vues pour le relief

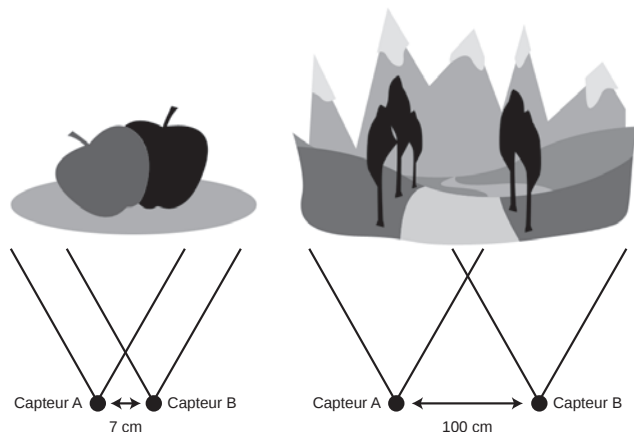
Pour créer un affichage en relief avec un effet le plus précis possible, nous devons prendre en compte un certain nombre d'éléments. Dans le cadre de ce chapitre, nous en évoquons seulement les grands principes. Pour connaître les techniques de création d'images en relief de manière détaillée, consultez le site de David Romeuf : <http://www.david-romeuf.fr>, très largement documenté sur le sujet.

Pour vous aider à considérer toutefois certains principes élémentaires, inhérents à la création d'images en relief, voici quelques recommandations :

- Vous devez d'abord disposer de deux images du même sujet, mais vous devez décaler le point de vue de chacune par rapport à l'autre. Vous reconstituez ainsi le point de vue des yeux qui est à l'origine de la vision en relief.
- Lors de la prise de vue photographique, la distance entre les deux points de vue, pour reconstituer un tel effet, doit être directement proportionnelle à la distance entre l'obturateur et le sujet visé. Ainsi, si vous photographiez un sujet de près, la distance entre les deux points de vue doit être égale à la distance entre deux pupilles d'œil humain (6 à 10 cm). Si vous capturez, en revanche, un paysage lointain (une montagne, une architecture, un volume paysagé ou urbain), vous devez séparer les points de vue de plusieurs dizaines de centimètres, voir d'environ un mètre pour les grands paysages (voir Figure 12.1).

Figure 12.1

Distance entre les points de prise de vue.



Dans la conception d'une image anaglyphe pour le relief, nous devons également éviter de capturer des objets à dominante colorée proche de celle de l'une des deux lentilles. Un objet de couleur rouge, par exemple, ne pourrait être perçu par l'une des deux lentilles et son volume ne pourrait donc pas être reconstitué. Évitez, par conséquent, la mise en scène d'objets rouges et bleus, si vous travaillez avec la technique de l'anaglyphe,

Pour réaliser un anaglyphe, nous décalons d'abord les couches de couleurs rouge, vert et bleu, dans Photoshop. Une fois l'image exportée pour le Web, nous pouvons en contrôler l'affichage, dans Flash, avec ActionScript.

Réaliser un anaglyphe avec Photoshop

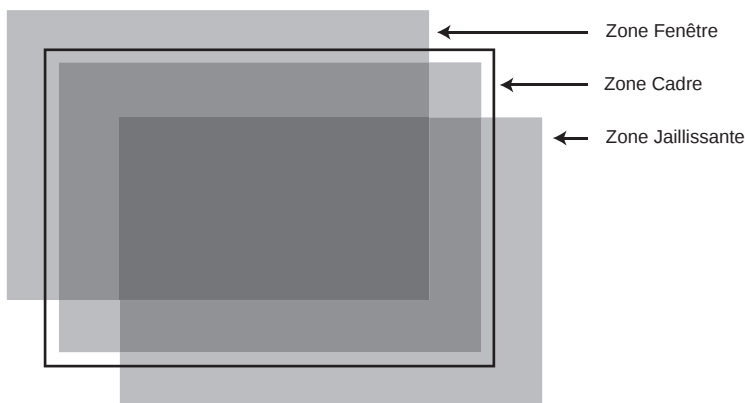
Dans cette section, nous allons d'abord comprendre le principe de l'image jaillissante ou fenêtre, à partir d'une image simple. Puis, nous verrons comment convertir deux prises de vue séparées, en un seul anaglyphe, à l'aide de Photoshop.

Le principe de l'image jaillissante ou fenêtre

Dans une image en relief, nous distinguons trois zones. La zone cadre, la zone jaillissante et la zone fenêtre (voir Figure 12.2).

Figure 12.2

Définition des zones d'affichage pour le relief.



La zone cadre représente la partie sans relief de l'image. La zone jaillissante est la partie qui se place devant le cadre, c'est-à-dire, la partie de l'image qui sort de l'écran, au premier plan. La partie fenêtre représente la partie située à l'arrière-plan du cadre, c'est-à-dire, la partie en retrait et située en profondeur de l'écran.

Le principe des trois zones est important, car selon la manière dont nous créons l'anaglyphe dans Photoshop, selon le type de décalage apporté à la couche de couleur rouge, nous pouvons choisir si l'image sera placée en retrait (fenêtre) ou en avant (jaillissante) (voir Figure 12.3 et 12.4).

Figure 12.3

Anaglyphe
fenêtre.

**Figure 12.4**

Anaglyphe
jaillissant.



Dans cette section, nous allons convertir une image initialement "plate", en image fenêtre, puis en image jaillissante. Il n'est pas nécessaire de disposer de deux images spécifiques pour le relief pour effectuer cette manipulation. Toute l'image sera simplement projetée dans son intégralité, soit vers l'arrière, soit vers l'avant. Sur ce principe, nous pouvons donc déjà imaginer pouvoir composer assez simplement, avec Photoshop, des images en relief à partir d'objets 2D détournés, sur des plans dissociés. Nous abordons cette technique un peu plus loin dans ce chapitre.

La zone fenêtre

Dans cette section, nous allons voir comment réaliser une image de type fenêtre.



Exemples > relief > Lacoste.jpg

1. Lancez Photoshop.
2. Ouvrez le document "Lacoste.jpg", situé dans le dossier "relief" des exemples du livre. Cette image est normale et n'affiche pas de relief. Elle est en RVB et son unique calque est aplati (voir Figure 12.5).

Figure 12.5

Aperçu de l'image
Lacoste.jpg.



3. Affichez la fenêtre Couches en déroulant le menu Fenêtre > Couches (voir Figure 12.6).

Figure 12.6

Fenêtre Couches.



La fenêtre affiche un aperçu des composantes colorimétriques de l'image, à travers différentes couches RVB. La couche RVB affiche les composantes de l'image dans sa globalité. Les couches R, V et B affichent respectivement les couleurs rouge, vert et bleu de l'image. Vous pouvez cliquer sur chacune d'entre elles pour en distinguer les propriétés.

Nous allons déplacer la couche Rouge à gauche, d'une dizaine de pixels, pour faire sortir l'image de la zone cadre et créer un effet Fenêtre.

1. Cliquez sur la couche Rouge. Elle est active. Les autres couches sont masquées. Dans l'image, la luminosité des valeurs de rouge est matérialisée en noir et blanc (voir Figure 12.7).

Figure 12.7

Activation de la couche Rouge.



2. Pour déplacer la couche de 10 pixels vers la gauche, utilisez l'outil de déplacement (raccourci V). Sélectionnez tous les pixels de la couche en faisant Ctrl+A (Windows) ou Cmd+A (Mac).
3. Puis, appuyez simultanément sur la touche Maj+Flèche gauche du clavier. La couche Rouge est aussitôt déplacée de 10 pixels vers la gauche.
4. Désactivez la sélection en faisant Ctrl+D (Windows) ou Cmd+D (Mac).
5. Cliquez à nouveau sur la couche intitulée RVB pour afficher l'intégralité de l'image. L'image en couleur affiche désormais un décalage entre le rouge et les composantes vert et bleu. L'image obtenue est un anaglyphe (voir Figure 12.8).

Figure 12.8

Aperçu
de l'anaglyphe.



6. Recadrez éventuellement l'image pour supprimer la marge située à droite.
7. Puis, observez l'image à l'aide de lunettes rouge et bleue, en prenant soin de placer le filtre rouge sur l'œil gauche.

Vous pouvez maintenant voir une image projetée à l'intérieur de l'écran, dans la zone fenêtre.

La zone jaillissante

Dans cette section, nous allons voir comment réaliser une image de type jaillissante.



Exemples > relief > Lacoste-chien.psd

Le principe pour créer un volume jaillissant est identique à celui utilisé pour le mode Fenêtre, sauf que nous déplaçons la couche Rouge vers la droite afin de projeter l'image en premier plan. Notez cependant que, parce que notre cerveau les analyse ainsi, l'effet jaillissant ne fonctionne qu'avec des éléments qui représentent des formes en volume, et non en profondeur comme cette ruelle représentée ici. Pour que l'effet fonctionne en mode jaillissant, nous utilisons donc, dans notre exemple, un sujet détourné qui sera positionné au premier plan.

1. Ouvrez le fichier Lacoste-chien.psd (voir Figure 12.9).

Figure 12.9

Aperçu du document
Lacoste-
chien.psd.



2. Activez le calque du chien.
3. Directement, dans la fenêtre Couches, cliquez sur la couche Rouge.
4. Sélectionnez tous les pixels (Ctrl+A pour Windows ou Cmd+A pour Mac) et déplacez-les de 10 pixels vers la droite à l'aide de l'outil Flèche et déplacement (raccourci V). Vous remarquerez que le déplacement génère une frange blanche de 10 pixels à gauche du sujet (voir Figure 12.10).

Figure 12.10

Frange à gauche
du sujet.



Nous pouvons la supprimer en gommant directement sur toutes les couches du calque chien en même temps.

5. Intervertissez la sélection encore active en faisant Ctrl+Maj+i (Windows) ou Cmd+Maj+i (Mac).
6. Cliquez d'abord sur la couche RVB pour atteindre l'ensemble des composantes colorimétriques du calque.
7. Sélectionnez l'outil Gomme – raccourci E, comme *Erase* qui signifie gommer en anglais – (voir Figure 12.11).

Figure 12.11

Sélection de l'outil Gomme.



8. Puis, gomez la partie gauche du calque chien, matérialisée à présent en rouge (voir Figure 12.12).

Figure 12.12

Gommage de la frange à gauche.



9. Une fois la frange supprimée, désactivez la sélection en faisant Ctrl+D (Windows) ou Cmd+D (Mac).
10. Reprenez les lunettes rouge et bleue. Placez-vous à plus d'un mètre de votre écran. Le chien semble sortir de l'écran tandis que la ruelle se prolonge en profondeur (voir Figure 12.13).

Figure 12.13

Reconstitution du résultat obtenu.



Cette composition est affichée en relief, bien que chaque sujet ne soit pas capturé en relief. Seuls les plans sur lesquels sont disposés les objets sont en relief. Pour réaliser un véritable anaglyphe en relief, vous devez utiliser deux images qui représentent le même sujet, mais pris à quelques centimètres de décalage.

Créer l'anaglyphe à partir de deux images

Nous abordons ici la création de l'anaglyphe, dans Photoshop, à partir de deux images prises sur le même sujet avec un intervalle de 10 cm.



Exemples > relief > relief.psd

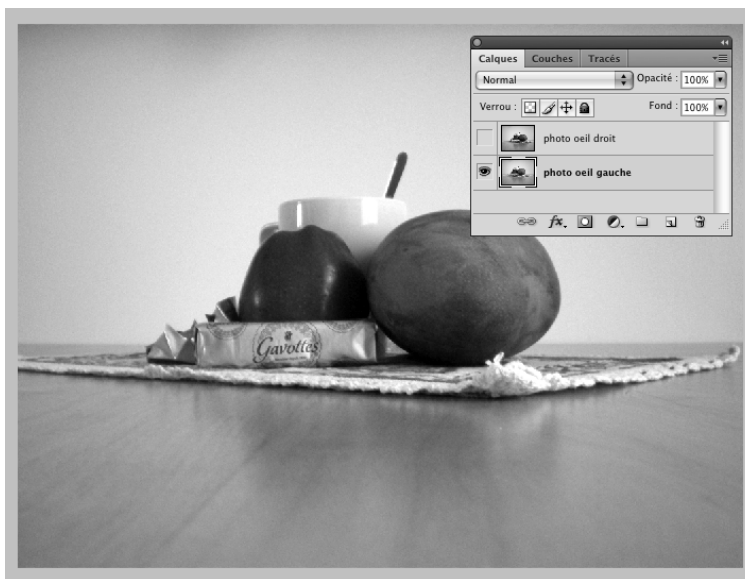
Ouvrez le document "relief.psd". La composition représente quelques fruits disposés autour d'un mug à thé. La fenêtre des calques (Fenêtre > Calques) affiche deux images. Le calque du haut représente une prise de vue effectuée au niveau de l'œil droit (voir Figure 12.14). Celui du bas représente le point de vue de l'œil gauche (voir Figure 12.15).

Figure 12.14

Aperçu du calque œil droit.

**Figure 12.15**

Aperçu du calque œil gauche.



Pour réaliser un anaglyphe à partir de deux images, nous plaçons les informations de la couche Rouge d'une première image à la place des informations de la couche Rouge de la seconde :

1. Affichez la fenêtre Couches et cliquez directement sur la couche Rouge. Puis, sélectionnez tous les pixels de la couche en faisant Ctrl+A sous Windows, ou Cmd+A sous Mac (voir Figure 12.16).

Figure 12.16

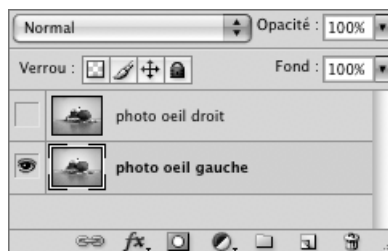
Sélection de la couche Rouge.



2. Faites Ctrl+C (Windows) ou Cmd+C (Mac) pour copier la couche Rouge.
3. Revenez dans la fenêtre des calques et activez maintenant l'autre calque "photo œil gauche". Pour voir ce calque placé, masquez le calque précédent, situé au sommet la fenêtre des calques (voir Figure 12.17).

Figure 12.17

Isolement du calque œil gauche.



4. Revenez dans la fenêtre Couches et cliquez sur la couche Rouge pour l'activer.
5. Sélectionnez toute la couche en faisant Ctrl+A (Windows) ou Cmd+A (Mac).
6. Puis, collez les informations Rouges du calque précédent en faisant Ctrl+V (Windows) ou Cmd+V (Mac).
7. Activez enfin la couche globale RVB de ce calque pour en visualiser le rendu (voir Figure 12.18).

La nouvelle couche Rouge est bien appliquée au document. Mais elle est trop décalée par rapport à l'image de gauche. Pour que le relief prenne, nous devons rapprocher les éléments. À l'aide de l'outil de déplacement (raccourci V), déplacez la couche Rouge de 10 ou 20 pixels pour la recentrer (voir Figure 12.19).

Figure 12.18

Placement de la nouvelle couche Rouge.

**Figure 12.19**

Recentrage de la couche Rouge.



L'image est prête à être recadrée. Elle mesure $1\,024 \times 768$ pixels. Nous allons la recadrer afin qu'elle occupe les mêmes dimensions que celles de notre document Flash :

8. Activez l'outil de recadrage – raccourci C, comme *Crop* qui signifie rogner en anglais – (voir Figure 12.20).

Figure 12.20

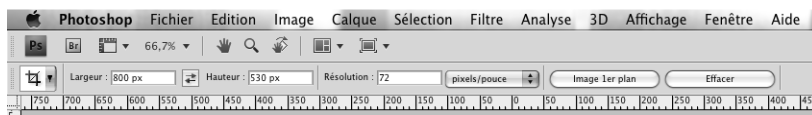
Sélection de l'outil de recadrage.



9. Dans les options, situées au sommet de l'interface de Photoshop, sous la barre de menu, inscrivez les valeurs : 800px, 530px et 72dpi (voir Figure 12.21).

Figure 12.21

Options de recadrage.



10. Recadrez l'image en centrant approximativement le sujet. Puis validez en cliquant sur Entrée (voir Figure 12.22).

Figure 12.22

Aperçu de la zone de recadrage.



11. Une fois l'image recadrée, vous pouvez aussi l'aplatir. Faites Calques > Aplatir l'image.
12. Puis exportez pour le Web en faisant Fichier > Enregistrer pour le Web et les périphériques. Choisissez une option de compression JPEG de qualité supérieure ou égale à 60 (voir Figure 12.23). Puis confirmez l'enregistrement.

Figure 12.23

Aperçu de la fenêtre d'enregistrement pour le Web.



L'image est prête à être importée dans Flash.

Gérer un anaglyphe en ActionScript

L'intégration d'un anaglyphe dans Flash est relativement simple puisqu'il suffit d'importer l'image dans la scène. Mais il convient aussi de bien situer le type d'image dont nous disposons selon le contexte d'exécution. Si vous affichez une image jaillissante, il faut naturellement la placer de préférence au sommet de la liste d'affichage ou sur un calque placé au-dessus des autres. Inversement, si l'image est de type fenêtre, vous la placerez à l'arrière des autres objets de la scène.

Cela signifie aussi que pour une image jaillissante, dans un contexte de scène en 3D, vous lui affecterez un index z paradoxalement inférieur à 0. Pour un type fenêtre, vous appliquerez un index z supérieur à 0. Dans Flash, nous rappelons le z augmente en allant vers le fond, tandis qu'il prend une valeur négative dès qu'il se rapproche de l'écran. Plus l'objet est proche de l'écran et au premier plan, plus son index z diminue.



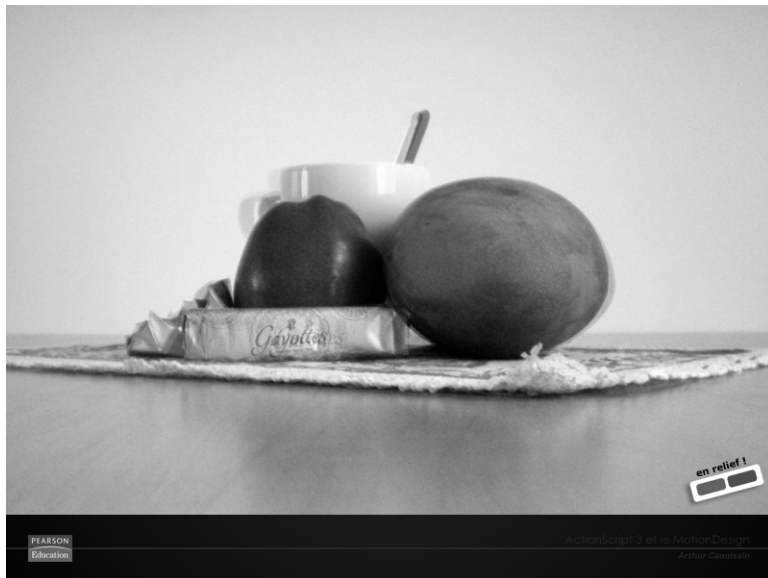
Exemples > ch11_vraiRelief_1 fla

Dans le document "ch11_vrairelief_1 fla", sur la scène, nous pouvons voir un MovieClip qui contient deux images. Une image classique est placée sur l'image 1 du scénario. La version en relief est placée sur l'image 2. Au-dessus, sur la scène principale, un bouton en forme de lunette 3D permet de basculer de l'une à l'autre.

En publiant le document, quand nous cliquons sur les lunettes, l'image en relief remplace effectivement l'image normale (voir Figure 12.24).

Figure 12.24

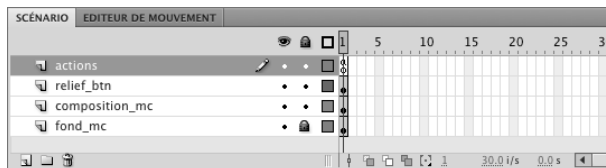
Aperçu du document.



Dans la fenêtre de scénario, au-dessus du calque Actions, nous distinguons le MovieClip `composition_mc` du bouton `relief_btn` (voir Figure 12.25).

Figure 12.25

Aperçu de la fenêtre de scénario.



Dans la fenêtre Actions, apparaît le code suivant :

```
composition_mc.stop();
//
relief_btn.addEventListener(MouseEvent.CLICK,basculerEnRelief);
function basculerEnRelief (evt:MouseEvent) {
    if (composition_mc.currentFrame==1) {
        composition_mc.nextFrame();
    }else {
        composition_mc.prevFrame();
    }
}
```

Dans ce programme, nous plaçons un stop au début du MovieClip qui contient les deux images afin d'éviter qu'une animation ne se produise.

Plus bas, un écouteur, attaché au bouton `relief_btn`, lance le changement d'image en fonction de celle qui est active. L'image active est détectée, elle, grâce à la méthode `currentFrame()`.

Dans cette condition, si l'image active est la première, nous basculons vers la seconde avec la méthode `nextFrame()` qui désigne l'image suivante. Nous revenons, en revanche, à la première image si c'est la seconde qui est identifiée, avec la méthode `prevFrame()` qui désigne l'image précédente.



La vidéo en relief. La technique du relief peut également être appliquée à un flux vidéo. Pour cela, vous devez disposer soit d'une composition vidéo à plusieurs calques et déplacer la couche Rouge de chaque calque à gauche ou à droite en fonction de l'effet souhaité (avec After Effects ou Motion par exemple). Soit vous devez partir de deux flux vidéo distincts, représentant le point de visé des deux yeux, comme pour des images fixes, et fusionner le tout en un seul flux en prenant la couche rouge de l'un que vous placez à la place de la couche rouge de l'autre.

Dans After Effects, utilisez l'effet **Perspective > Lunette 3D** pour regrouper les flux vidéo. Dans les réglages d'effets, source G désigne le point de vue de l'œil gauche, source D, le point de vue de l'œil droit. Balance couleur Rouge Bleu signifie vision 3D. Ajustez le Décalage de convergence (normalement, si les vidéos sont bien tournées, la valeur est proche de zéro). Ajustez la Balance en fonction de la densité des lunettes (mesurable chez votre opticien). Puis, exportez la vidéo obtenue en F4V ou en FLV et intégrez-la simplement avec un composant `FLVPlayback`, comme vu au Chapitre 7.

À retenir

- Pour créer une image en relief, nous devons reproduire la vision humaine en capturant deux images dont les points de visé sont distants, l'un de l'autre, d'environ 6,5 cm pour un sujet proche, à près d'1 mètre pour un grand paysage.
- Nous devons copier la couche Rouge d'une image pour remplacer celle de l'autre.
- L'image devient de type fenêtre dès lors que la couche Rouge est située à gauche.
- L'image devient jaillissante dès que la couche Rouge est placée sur la droite.
- Il est possible de réaliser des images en relief à partir d'images classiques. Pour cela, nous détournons chaque sujet et décalons la position de sa couche par rapport aux couches Rouge des autres calques de la composition.
- Il est possible, en `ActionScript`, de détecter la position courante d'une image de scénario, grâce à la méthode `currentTarget()`.

Interface SWF en relief dynamique

Nous venons de voir comment se constitue un contenu en relief. Au chapitre précédent, nous avons vu comment agir sur les couches RVB et Alpha d'une image, dynamiquement. Précédemment encore, nous avons abordé la gestion de la 3D avec l'index `z`. En combinant toutes ces techniques, nous pouvons construire une interface 3D en relief dynamique.

Le principe de l'interface en relief dynamique est de permettre le positionnement d'objets (textes, images, formes graphiques) dans un conteneur et de leur attribuer, pour chacun d'entre eux, un index z. En publiant le document, les couches Rouge, Vert et Bleu sont automatiquement séparées en fonction de la profondeur de z.

Dans cette section, nous allons étudier le mécanisme d'un moteur de rendu d'objets en vrai relief.

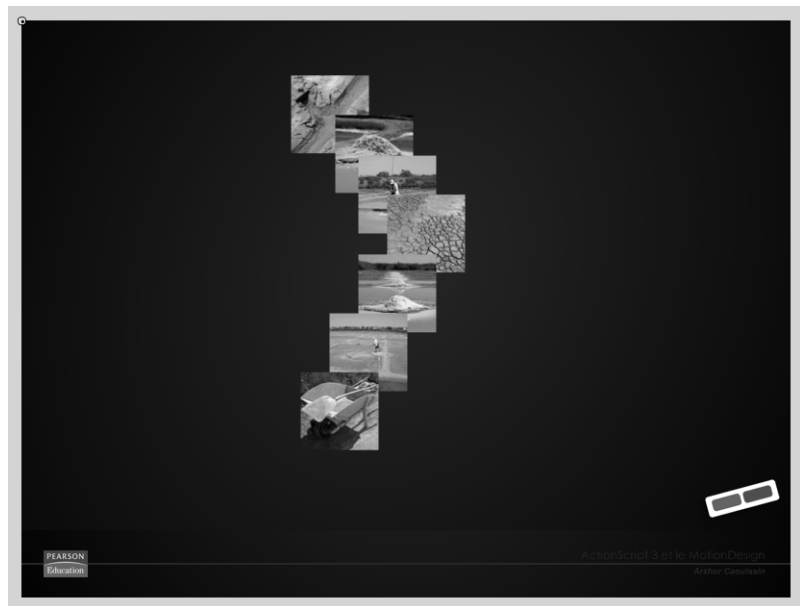


Exemples > ch12_vraiRelief_2 fla

Dans le document "ch12_vrairelief_2 fla", sur la scène principale, apparaît un MovieClip `composition_mc`, à l'intérieur duquel sont positionnés des symboles. Ces symboles sont répartis de sorte que celui qui doit apparaître au premier plan est situé naturellement au-dessus des autres. Dans le code du calque Actions, que nous allons voir, un index z est également attribué pour chacun d'entre eux (voir Figure 12.26). N'oubliez pas que pour qu'un objet puisse disposer d'un index z, celui-ci doit être un MovieClip.

Figure 12.26

Aperçu de la scène principale.



Lorsque le document est publié, si le bouton `relief_btn` situé à droite de l'écran est activé, la composition bascule presque instantanément en vrai relief. Le décalage de la couche Rouge est directement proportionnel à la position de l'objet sur l'index z (voir Figure 12.27).

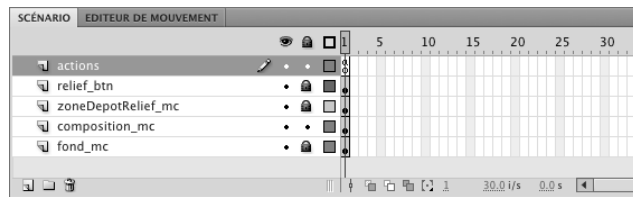
Dans la fenêtre de scénario du document, au-dessus du calque `fond_mc`, sont répartis : le symbole `composition_mc`, qui affiche tous les éléments à projeter en relief ; au-dessus, le symbole `zoneDepotRelief_mc`, qui contient les objets utilisés par le moteur de relief ; ainsi que le bouton `relief_btn` (voir Figure 12.28).

Figure 12.27

Aperçu du document publié, avec l'option relief activée.

**Figure 12.28**

Scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//----- personalisation

var echelleZ:Number=400;
var echelleRelief:Number=0.015;
//
function indexZ() {
    // toujours <0
    composition_mc.p1_mc.z=-700;
    composition_mc.p2_mc.z=-600;
    composition_mc.p3_mc.z=-500;
    composition_mc.p4_mc.z=-400;
    composition_mc.p5_mc.z=-300;
    composition_mc.p6_mc.z=-200;
    composition_mc.p7_mc.z=-100;
}

relief_btn.addEventListener(MouseEvent.CLICK,basculerEnRelief);
function basculerEnRelief(evt:MouseEvent) {
    composition_mc.visible=false;
    moteurRelief();
}
```

```
//
+++++
//+++++ MOTEUR RELIEF (Arzhur CAOUISSIN) +++++
//
+++++

// tableauRouge
var tableauRouge:Array = new Array();
var filtreRouge:ColorMatrixFilter=new ColorMatrixFilter(tableauRouge);
tableauRouge = new Array();
tableauRouge=tableauRouge.concat([-1,0,0,0,255]);
tableauRouge=tableauRouge.concat([0,-1,0,0,0]);
tableauRouge=tableauRouge.concat([0,0,-1,0,0]);
tableauRouge=tableauRouge.concat([0,0,0,1,0]);
filtreRouge=new ColorMatrixFilter(tableauRouge);
// tableauVert
var tableauVert:Array = new Array();
var filtreVert:ColorMatrixFilter=new ColorMatrixFilter(tableauVert);
tableauVert = new Array();
tableauVert=tableauVert.concat([-1,0,0,0,0]);
tableauVert=tableauVert.concat([0,-1,0,0,255]);
tableauVert=tableauVert.concat([0,0,-1,0,0]);
tableauVert=tableauVert.concat([0,0,0,1,0]);
filtreVert=new ColorMatrixFilter(tableauVert);
// tableauBleu
var tableauBleu:Array = new Array();
var filtreBleu:ColorMatrixFilter=new ColorMatrixFilter(tableauBleu);
tableauBleu = new Array();
tableauBleu=tableauBleu.concat([-1,0,0,0,0]);
tableauBleu=tableauBleu.concat([0,-1,0,0,0]);
tableauBleu=tableauBleu.concat([0,0,-1,0,255]);
tableauBleu=tableauBleu.concat([0,0,0,1,0]);
filtreBleu=new ColorMatrixFilter(tableauBleu);
// tableauNoir
var tableauNoir:Array = new Array();
var filtreNoir:ColorMatrixFilter=new ColorMatrixFilter(tableauNoir);
tableauNoir = new Array();
tableauNoir=tableauNoir.concat([-1,0,0,0,0]);
tableauNoir=tableauNoir.concat([0,-1,0,0,0]);
tableauNoir=tableauNoir.concat([0,0,-1,0,0]);
tableauNoir=tableauNoir.concat([0,0,0,1,0]);
filtreNoir=new ColorMatrixFilter(tableauNoir);
//+++++

var largeurDonnees:Array=new Array();
var hauteurDonnees:Array=new Array();

function moteurRelief() {
    for (var i:Number=0; i<composition_mc.numChildren; i++) {
        largeurDonnees.push(composition_mc.getChildAt(i).width);
        hauteurDonnees.push(composition_mc.getChildAt(i).height);
    }
    //
    if (i==composition_mc.numChildren-1) {
        var boucle:Timer=new Timer(500,1);
        boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
        boucle.start();
        indexZ();
    }
}
```

```

    }
  }
}

function lancerBoucle(evt:TimerEvent) {
    affichageRelief();
}

function affichageRelief() {
    for (var i:Number=0; i<composition_mc.numChildren; i++) {
        //
        var enveloppeImage:BitmapData=new BitmapData(800,600,false,0x00000000);
        enveloppeImage.draw(composition_mc.getChildAt(i));
        var pixelsImage:Bitmap=new Bitmap(enveloppeImage);
        var capture:BitmapData=pixelsImage.bitmapData;
        //
        var bitmapR:BitmapData=new BitmapData(largeurDonnees[i],
        ➤ hauteurDonnees[i],true,0x000000);
        var bitmapV:BitmapData=new BitmapData(largeurDonnees[i],
        ➤ hauteurDonnees[i],true,0x000000);
        var bitmapB:BitmapData=new BitmapData(largeurDonnees[i],
        ➤ hauteurDonnees[i],true,0x000000);
        var bitmapA:BitmapData=new BitmapData(largeurDonnees[i],
        ➤ hauteurDonnees[i],true,0x000000);
        //
        var imageR:Bitmap=new Bitmap(bitmapR);
        var imageV:Bitmap=new Bitmap(bitmapV);
        var imageB:Bitmap=new Bitmap(bitmapB);
        var imageA:Bitmap=new Bitmap(bitmapA);
        //
        bitmapR.copyChannel(capture, capture.rect, new Point(0,0),
        ➤ BitmapDataChannel.RED, BitmapDataChannel.ALPHA);
        bitmapV.copyChannel(capture, capture.rect, new Point(0,0),
        ➤ BitmapDataChannel.GREEN, BitmapDataChannel.ALPHA);
        bitmapB.copyChannel(capture, capture.rect, new Point(0,0),
        ➤ BitmapDataChannel.BLUE, BitmapDataChannel.ALPHA);
        bitmapA.copyChannel(capture, capture.rect, new Point(0,0),
        ➤ BitmapDataChannel.ALPHA, BitmapDataChannel.ALPHA);
        //
        imageR.filters=[filtreRouge];
        imageV.filters=[filtreVert];
        imageB.filters=[filtreBleu];
        imageA.filters=[filtreNoir];
        //
        var AffichageRelief:MovieClip=new ZoneAffichageRelief();
        zoneDepotRelief_mc.addChildAt(AffichageRelief,i);
        // la couche RVB
        AffichageRelief.coucheAlpha_mc.addChild(imageA);
        imageA.x=composition_mc.getChildAt(i).x;
        imageA.y=composition_mc.getChildAt(i).y;
        // la couche Rouge
        AffichageRelief.coucheRouge_mc.addChild(imageR);
        imageR.x=composition_mc.getChildAt(i).x+(composition_mc.getChildAt(i).z)
        ➤ *echelleRelief;
    }
}

```

```

        imageR.y=composition_mc.getChildAt(i).y;
        // la couche Bleu
        AffichageRelief.coucheVerte_mc.addChild(imageV);
        imageV.x=composition_mc.getChildAt(i).x;
        imageV.y=composition_mc.getChildAt(i).y;
        // la couche Bleu
        AffichageRelief.coucheBleue_mc.addChild(imageB);
        imageB.x=composition_mc.getChildAt(i).x;
        imageB.y=composition_mc.getChildAt(i).y;
        //
        imageR.scaleX=(composition_mc.getChildAt(i).z)/echelleZ;
        imageR.scaleY=(composition_mc.getChildAt(i).z)/echelleZ;
        imageV.scaleX=(composition_mc.getChildAt(i).z)/echelleZ;
        imageV.scaleY=(composition_mc.getChildAt(i).z)/echelleZ;
        imageB.scaleX=(composition_mc.getChildAt(i).z)/echelleZ;
        imageB.scaleY=(composition_mc.getChildAt(i).z)/echelleZ;
        imageA.scaleX=(composition_mc.getChildAt(i).z)/echelleZ;
        imageA.scaleY=(composition_mc.getChildAt(i).z)/echelleZ;
    }
}
//+++++
//+++++
//
+++++

```

Le système est scindé en deux parties. La première donne accès aux éléments personnalisables directement en rapport avec les objets disposés dans la scène. La seconde constitue le moteur de rendu en relief.

Dans la première partie, nous définissons d'abord les coefficients de déformation utilisés par le moteur :

```

var echelleZ:Number=400;
var echelleRelief:Number=0.015;

```

Le moteur fonctionne de la manière suivante. Nous plaçons des objets dans l'espace 3D à l'intérieur du symbole `composition_mc`. L'image projetée par ce conteneur est alors copiée en ActionScript, puis traitée, avant d'être redistribuée sur un autre conteneur : `zoneDepotRelief_mc`. Pendant le traitement, les couches sont séparées et l'une d'entre elles, la Rouge, est décalée.

Une fois le traitement terminé, les couches sont distribuées dans un MovieClip placé dynamiquement en ActionScript, à partir d'une occurrence exportée de la bibliothèque. Cette occurrence contient quatre MovieClip destinés respectivement pour chacune des quatre composantes de couleur Rouge, Vert, Bleu et Alpha.

Afin que les couches colorimétriques fusionnent correctement, nous avons appliqué, sur les deux premiers MovieClip, une propriété d'affichage de type Ajout (c'est un mode de fusion accessible depuis l'inspecteur de propriétés). Ainsi, lorsque les trois couches RVB sont superposées, sans décalage, l'image est entièrement reconstituée. Mais, pour reconstituer l'effet de relief, nous conservons naturellement le décalage de la couche Rouge.

Le symbole `AffichageRelief`, disponible dans la bibliothèque, est exporté pour ActionScript avec le nom de classe `ZoneAffichageRelief`. C'est lui qui sera distribué dans le symbole `zoneDepotRelief_mc`, autant de fois que d'objets à reconstituer.

Exporter un symbole pour ActionScript

Pour exporter un symbole pour ActionScript en vue de réaliser un interfaçage dynamique, dans la bibliothèque (Ctrl+L pour Windows ou Cmd+L pour Mac), directement sur l'objet à exporter, faites clic-droit > Propriétés. Puis, dans la boîte de dialogue, cochez la case Exporter pour ActionScript. Attribuez ensuite un nom de classe à l'objet sans caractère spécial, ni accent, espace ou signe particulier. En validant, le player fabriquera pour nous cette classe à la volée si celle-ci n'existe pas.

La variable `echelleZ` permet de retranscrire l'échelle de déformation des couches, selon l'index `z` attribué à chaque objet :

```
var echelleZ:Number=400;
```

Ce coefficient est utilisé pour recréer la profondeur, car en réalité, dans notre programme, les propriétés 3D que nous avons affectées aux objets de la scène ne sont pas prises en compte lorsque le calcul d'affichage en relief est appliqué. Elle n'est appliquée qu'une fois les couches séparées. Les propriétés `width` et `height` utilisées, perçoivent, en effet, des valeurs différentes lorsque deux couches alpha de mêmes dimensions sont placées à un index `z` distinct. Une incohérence que nous contournons en activant l'affichage 3D uniquement lorsque le calcul de séparation des couches est terminé. Sans quoi les couches alpha de chaque objet auraient été bien plus grandes que les objets eux-mêmes.

La deuxième variable, `echelleRelief`, est employée pour définir la valeur de décalage de la couche Rouge :

```
var echelleRelief:Number=0.015;
```

Plus cette valeur est élevée, plus le décalage de la couche Rouge sera marqué.

À la suite, une fonction détermine l'index `z` de chaque objet placé dans le symbole `composition_mc` :

```
function indexZ() {  
    // toujours <0  
    composition_mc.p1_mc.z=-700;  
    composition_mc.p2_mc.z=-600;  
    composition_mc.p3_mc.z=-500;  
    composition_mc.p4_mc.z=-400;  
    composition_mc.p5_mc.z=-300;  
    composition_mc.p6_mc.z=-200;  
    composition_mc.p7_mc.z=-100;  
}
```

Le premier objet de la liste d'affichage, celui placé en premier plan dans le symbole `composition_mc`, affiche un index le plus proche de l'écran (-700). Les autres se rapprochent progressivement de la valeur 0. Ces valeurs sont isolées dans une fonction de manière à permettre au moteur de capturer les dimensions des objets avant de modifier leur index `z`, juste avant de lancer le mode d'affichage en relief. Afin de préserver les proportions d'échelle des objets en fonction de leur index `z`, nous limitons la gestion des objets à un index inférieur à 0, et donc, à un relief jaillissant.

Ensuite, un gestionnaire d'événements appelle le moteur de rendu, sur action de l'utilisateur :

```
relief_btn.addEventListener(MouseEvent.CLICK,basculerEnRelief);
function basculerEnRelief(evt:MouseEvent) {
    composition_mc.visible=false;
    moteurRelief();
}
```

Nous masquons d'abord le symbole `composition_mc` qui affiche les objets dans un espace 2D. Puis, nous activons la fonction `moteurRelief()` qui va placer, dans le clip vide `zoneDepotrelief_mc`, situé au premier plan, les objets d'affichage en relief.

Intervient ensuite le moteur de rendu. Le moteur génère, pour commencer, autant de filtres colorimétriques que de couches à isoler. Dans ce système, nous utilisons quatre couches. La couche Rouge permet de reconstituer le relief en le décalant d'une valeur proportionnelle à l'index `z` des objets. Les couches Vert et le Bleu sont séparées, mais resteront superposées pour recréer la base de l'image. L'Alpha, enfin, va permettre d'éviter que les couches, même superposées, apparaissent légèrement translucides. Souvenez-vous, nous avons appliqué un mode de fusion par Ajout pour les couches Roug et Vert. Or, la couche Bleu n'est pas nécessairement entièrement opaque. Cela dépend de l'image. Si nous n'ajoutons pas une couche supplémentaire qui reprenne l'ensemble des informations opaques de l'image, nous obtiendrons des images partiellement translucides.

Dans un premier temps, nous définissons donc les filtres de couleurs que nous appliquons plus loin, à chaque couche séparée :

```
// tableauRouge
var tableauRouge:Array = new Array();
var filtreRouge:ColorMatrixFilter=new ColorMatrixFilter(tableauRouge);
tableauRouge = new Array();
tableauRouge=tableauRouge.concat([-1,0,0,0,255]);
tableauRouge=tableauRouge.concat([0,-1,0,0,0]);
tableauRouge=tableauRouge.concat([0,0,-1,0,0]);
tableauRouge=tableauRouge.concat([0,0,0,1,0]);
filtreRouge=new ColorMatrixFilter(tableauRouge);
```

Nous déclinons le principe pour chaque couche. À la suite, nous créons un tableau (Array) qui enregistre les dimensions des images, avant qu'elles ne soient projetées en relief. Le premier tableau stocke les largeurs et le second, les hauteurs :

```
var largeurDonnees:Array=new Array();
var hauteurDonnees:Array=new Array();
```



Mécanisme d'un tableau (Array). Rappel : un tableau fonctionne un peu comme la définition d'une variable, à ceci près que la structure peut implémenter un nombre de valeurs indéterminées. Pour ajouter une valeur dans un tableau, nous employons la méthode `push()`. En paramètre de cette méthode, nous spécifions la valeur à ajouter.

Dans notre exemple, les deux tableaux sont instanciés en amont. C'est à travers une boucle `for` que nous récupérons les valeurs. Cette boucle permet d'exécuter autant d'enregistrements que le symbole `composition_mc` affiche d'objets (en fonction de `numChildren`) :

```
function moteurRelief() {
    for (var i:Number=0; i<composition_mc.numChildren; i++) {
```

```

        largeurDonnees.push(composition_mc.getChildAt(i).width);
        hauteurDonnees.push(composition_mc.getChildAt(i).height);
        //
        if (i==composition_mc.numChildren-1) {
            var boucle:Timer=new Timer(500,1);
            boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
            boucle.start();
            indexZ();
        }
    }
}

function lancerBoucle(evt:TimerEvent) {
    affichageRelief();
}

```

Dès que la boucle est terminée ($i == \text{composition_mc.numChildren} - 1$), nous activons le moteur de rendu avec l'appel de la fonction `affichageRelief()`. Dans ce dispositif, nous devons attendre que la boucle soit terminée, car le moteur utilise les données enregistrées par la boucle pour effectuer le calcul d'affichage. Si nous lançons le moteur avant d'avoir défini les valeurs nécessaires, celui-ci ne pourrait fonctionner et retournerait une erreur. Pour garantir le délai, nous ajoutons même un chronomètre qui active la fonction seulement 500 millisecondes après l'enregistrement, soit une demi-seconde après l'enregistrement. Pour nous assurer que le chronomètre se déclenche uniquement suite aux instructions lancées en début de boucle, nous le plaçons à l'intérieur de la boucle.

À l'intérieur du moteur de rendu (fonction `affichageRelief()`), plus bas dans le code, une autre boucle `for`, de même structure, instancie autant d'images que d'objets dans le conteneur :

```

var enveloppeImage:BitmapData=new BitmapData(800,600,false,0x00000000);
enveloppeImage.draw(composition_mc.getChildAt(i));

```

La méthode `BitmapData()` génère d'abord une surface d'affichage à quatre canaux (Alpha, Rouge, Vert et Bleu). Puis, la méthode `draw()` y place une capture de l'objet appelé en paramètre. La méthode `draw()` dessine, pour ainsi dire, sur l'objet `BitmapData`, ce qui est appelé en paramètre de la méthode `draw()`.

Ensuite, nous définissons chaque image capturée en tant qu'objet, afin de permettre de les redistribuer ensuite dans d'autres conteneurs et de leur appliquer des filtres et des propriétés :

```

var imageR:Bitmap=new Bitmap(bitmapR);
var imageV:Bitmap=new Bitmap(bitmapV);
var imageB:Bitmap=new Bitmap(bitmapB);
var imageA:Bitmap=new Bitmap(bitmapA);

```

À travers la méthode `copyChannel()`, nous récupérons la couche correspondant à une des quatre composantes colorimétriques. C'est, en somme, le noyau dur du moteur que nous définissons là :

```

bitmapR.copyChannel(capture, capture.rect, new Point(0,0),
    BitmapDataChannel.RED, BitmapDataChannel.ALPHA);

```

En paramètre de cette méthode, nous désignons, respectivement :

- L'objet à partir duquel nous voulons extraire une couche (capture).
- La surface que nous voulons extraire (c'est toujours un rectangle, ici, la propriété `rect` lit la surface de l'image avec `capture.rect`).
- Les coordonnées `x` et `y` du point d'origine de cette zone rectangulaire ; la couche composite concernée (RED, GREEN, BLUE ou ALPHA).
- La propriété opaque ou transparente de l'image. Dans le cas d'une transparence, elle est définie avec la propriété `BitmapDataChannel.ALPHA`.

Puis, nous appliquons les filtres définis en amont, sur chacune de ces couches, en fonction de leur teinte respective :

```
imageR.filters=[filtreRouge];
imageV.filters=[filtreVert];
imageB.filters=[filtreBleu];
imageA.filters=[filtreNoir];
```

Grâce à ces filtres, les couches qui étaient extraites (par défaut noires) adoptent à présent la teinte qui correspond à leur composante colorimétrique réelle.

Pour chaque objet, nous plaçons dynamiquement, dans le symbole `zoneDepotRelief_mc`, une instance du symbole `ZoneAffichageRelief` disponible dans la bibliothèque et exporté pour ActionScript :

```
var AffichageRelief:MovieClip=new ZoneAffichageRelief();
zoneDepotRelief_mc.addChildAt(AffichageRelief,i);
```

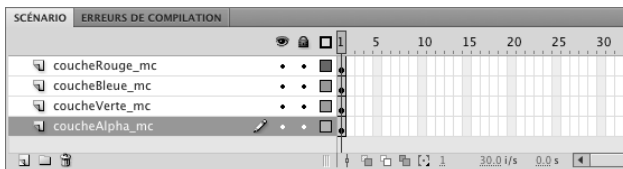
Pour chaque occurrence, nous l'ajoutons également à la liste d'affichage dans l'ordre qui correspond à l'empilement déjà défini pour le symbole `composition_mc` (`addChildAt(AffichageRelief,i)`).

Plus bas, nous affichons les occurrences dans le conteneur `zoneDepotRelief_mc`, en les distribuant en fonction de la couche véhiculée. Les couches Rouge et Vert sont placées dans un conteneur avec le mode de fusion par Ajout déjà actif. La couche Bleu est placée derrière. La couche Alpha, elle, se retrouve en dessous de toutes les autres et préserve l'opacité de chaque objet (voir Figure 12.29) :

```
// la couche RVB
AffichageRelief.coucheAlpha_mc.addChild(imageA);
imageA.x=composition_mc.getChildAt(i).x;
imageA.y=composition_mc.getChildAt(i).y;
```

Figure 12.29

Ordre d'affichage
des couches
Alpha, Rouge,
Vert et Bleu.



Nous en profitons pour ajuster aussi la position en `X`, principalement, pour la couche Rouge. Pour que l'effet de relief puisse être visible, nous devons effectivement décaler

l'abscisse du Rouge par rapport aux autres couches. C'est donc là que nous utilisons la variable `echelleRelief` qui nous aide à déterminer l'ampleur de ce décalage :

```
// la couche Rouge
AffichageRelief.coucheRouge_mc.addChild(imageR);
imageR.x=composition_mc.getChildAt(i).x+(composition_mc.getChildAt(i).z
↳ *echelleRelief;
imageR.y=composition_mc.getChildAt(i).y;
```

La position X de la couche Rouge est ainsi proportionnelle à la valeur du x de chaque objet.

Nous terminons la fonction en modifiant l'échelle `scaleX` et `scaleY` de chaque objet en fonction de son index x, pour recréer la déformation induite par la mise en relief des objets dans l'espace 3D. Rappelez-vous que l'affichage du rendu est déterminé à partir d'objets 2D, placés dans `composition_mc`, et, à ce titre, nous oblige à déformer le tout en fonction de leur profondeur, que nous avons attribuée ailleurs, pour conserver l'illusion de cette profondeur :

```
imageR.scaleX=(composition_mc.getChildAt(i).z)/-echelleZ;
imageR.scaleY=(composition_mc.getChildAt(i).z)/-echelleZ;
```

La variable `echelleZ`, également définie en amont du programme, permet de gérer la déformation pour obtenir un effet plus ou moins accentué de la perspective. Une valeur élevée diminue la déformation et donc, éloigne les points de fuite.

À retenir

- Il est possible d'extraire une couche de composante colorimétrique à l'aide de la méthode `copyChannel()`.
- Il est possible réaliser un site en relief dynamique à partir de propriétés distinctes de contenus 2D, s'ils sont répartis par exemple sur un index x. Le moteur de rendu en relief sépare les couches RVBA et les redistribue alors en fonction de la profondeur x des objets.
- L'ordre d'affichage des objets dans la composition est déterminant pour le réalisme du dispositif en relief.

Synthèse

Dans ce chapitre, vous avez vu comment créer des images pour le relief avec la technique des anaglyphes. Vous avez appris à les intégrer dans un document Flash. En décortiquant le mécanisme de fonctionnement de l'affichage en relief, et à l'aide des techniques abordées dans les précédents chapitres, vous avez également appris à créer un système d'affichage dynamique qui permette de convertir, automatiquement, tout objet situé dans un espace 2D ou 3D, en relief. Vous êtes en mesure de développer des systèmes d'affichage pour tout type de contenu dont la volumétrie apporte du sens (architecture, bijoux, œuvres d'art, design d'objets, produits high-tech, musées).

13

Les systèmes de navigation avancés

Introduction

Les boutons proposés par défaut dans Flash offrent des fonctionnalités souvent en deçà des besoins réels attendus en production. Par exemple, il est particulièrement difficile de contrôler, par ActionScript, des contenus distribués au sein de ce type d'objet. Ils ne proposent pas non plus d'état visité comme le ferait un simple hyperlien en HTML. En utilisant des symboles de type MovieClip, il est possible de réaliser des systèmes de navigation plus personnalisables, plus souples à mettre en forme, et plus originaux.

Dans ce chapitre, nous allons aborder l'utilisation de symboles de type MovieClip à travers différents dispositifs de navigation avancés.

Boutons MovieClip fixes avec état activé et visité

Dans cette première section, nous vous proposons de réaliser un menu avec différents boutons disposant, pour chacun d'eux, les états survolé (over), sortie (out), appuyé (down), relevé (up) et visité (visited).

ActionScript 3 nous permet de centraliser les interactions au sein d'un même gestionnaire grâce aux propriétés `target` et `currentTarget`. Nous les utilisons dans cet exemple pour optimiser notre code.

Nous présentons ici trois déclinaisons du même menu avec quelques variantes de codage. La première version exécute tous les états. La deuxième version, aussi, mais elle utilise une structure conditionnelle de type `Switch`, plus souple pour des menus longs. La troisième enfin aborde l'ergonomie en traitant les boutons en tant qu'éléments actifs et non plus simplement visités. Lorsqu'une nouvelle rubrique est affichée, les boutons des autres entrées sont restaurés.



Exemples > `ch13_systemesNavigation_1 fla`
Exemples > `ch13_systemesNavigation_1b fla`
Exemples > `ch13_systemesNavigation_1c fla`

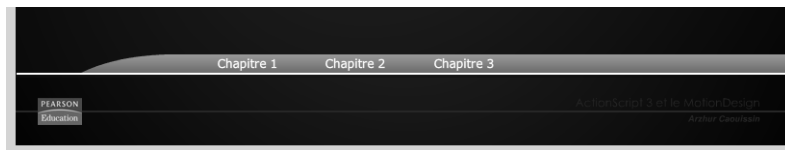
Les boutons visités

Dans le document "`ch13_systemesNavigation_1 fla`", la scène principale affiche un symbole MovieClip de nom d'occurrence `menu_mc` (voir Figure 13.1).

À l'intérieur de ce symbole sont répartis, vers les calques, trois boutons respectivement nommés `bouton1_mc`, `bouton2_mc` et `bouton3_mc`. Sont également visibles : un filet et quelques formes graphiques ajoutées pour l'habillage (voir Figure 13.2).

Figure 13.1

Aperçu de la scène principale.

**Figure 13.2**

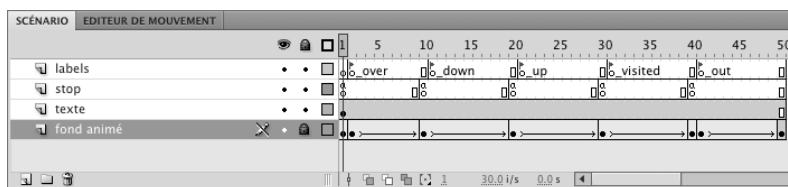
Aperçu du scénario du MovieClip menu_mc.



Dans le scénario de chacun de ces MovieClip, nous distinguons quatre calques dont certains affichent des interpolations (voir Figure 13.3).

Figure 13.3

Aperçu du scénario de chaque bouton MovieClip.

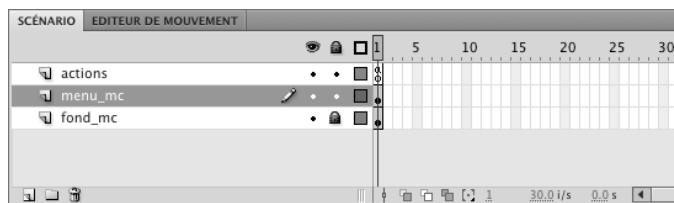


Le calque du bas contient une série d'animations placées les unes à la suite des autres. Le calque nommé texte affiche le libellé de chaque bouton. Le calque stop interrompt la tête de lecture avant chaque nouvelle animation, par la commande `stop()`. Enfin, le calque labels distribue une série d'étiquettes (*labels*) qui nous permettent d'identifier chaque animation en fonction de son rôle. Ces étiquettes permettent aussi de piloter par ActionScript le placement de la tête de lecture à ces endroits précis, lors du changement d'état survenu pour le bouton (état survolé, sortie, état appuyé, état relâché et état visité).

Dans la scène principale, au-dessus du calque `fond_mc` et `menu_mc`, apparaît le calque Actions (voir Figure 13.4).

Figure 13.4

Aperçu du scénario de la scène principale.



Dans le calque Actions, nous pouvons lire le code suivant :

```
//----- actions
//over
```

```

menu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    evt.target.gotoAndPlay("_over");
}
//down
menu_mc.addEventListener(MouseEvent.CLICK,down);
function down (evt:MouseEvent) {
    evt.target.gotoAndPlay("_down");
    evt.target.etatVisit=true;
    //
    if (evt.target.name=="bouton1_mc") {
        trace("action 1");
    }
    if (evt.target.name=="bouton2_mc") {
        trace("action 2");
    }
    if (evt.target.name=="bouton3_mc") {
        trace("action 3");
    }
}
//out
menu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    if (evt.target.etatVisit==true) {
        evt.target.gotoAndPlay("_visited");
    } else {
        evt.target.gotoAndPlay("_out");
    }
}
//up
menu_mc.addEventListener(MouseEvent.CLICK,up);
function up (evt:MouseEvent) {
    evt.target.gotoAndPlay("_up");
}

```

Pour appréhender le mécanisme du menu, nous devons d'abord comprendre que tous les boutons sont rassemblés à l'intérieur d'un symbole MovieClip nommé menu_mc. ActionScript 3 nous permet de cibler, grâce à la propriété target, chacun des objets qui capture l'événement, dans un même conteneur. Nous pouvons donc automatiser le processus d'interactivité avec plusieurs boutons réunis dans un clip, sans avoir à recréer de gestionnaire d'événements pour chacun d'entre eux.

Mais, nous souhaitons aussi exécuter un nombre d'événements parfois contradictoires. Par exemple, nous souhaitons qu'un bouton survolé affiche, en sortie, l'état par défaut. Mais, nous aimerions aussi que, si ce bouton était déjà activé, que ce soit un autre état qui apparaisse : l'état visité. Pour ce faire, nous utilisons une variable et une structure conditionnelle, qui, selon l'objet activé, va exécuter l'une ou l'autre des interpolations et ainsi permettre de résoudre cette pseudo-contradiction.

Dans le programme, nous créons un premier gestionnaire d'événements pour la définition de l'état survolé :

```

//over
menu_mc.addEventListener(MouseEvent.CLICK,over);

```



```
function over (evt:MouseEvent) {
    evt.target.gotoAndPlay("_over");
}
```

C'est bien le symbole `menu_mc` qui reçoit l'écouteur. Mais c'est bien l'occurrence de bouton survolée qui exécute la fonction (`evt.target`). Dans cette fonction, nous lisons l'interpolation placée à l'étiquette nommée `_over` lorsque le pointeur survole le bouton (`MOUSE_OVER`).

À la suite, un autre gestionnaire exécute une fonction lorsque le pointeur est enfoncé sur une occurrence de bouton (`MOUSE_DOWN`) :

```
//down
menu_mc.addEventListener(MouseEvent.CLICK,down);
function down (evt:MouseEvent) {
    evt.target.gotoAndPlay("_down");
    evt.target.etatVisit=true;
    //
    if (evt.target.name=="bouton1_mc") {
        trace("action 1");
    }
    if (evt.target.name=="bouton2_mc") {
        trace("action 2");
    }
    if (evt.target.name=="bouton3_mc") {
        trace("action 3");
    }
}
```

Dans cette fonction, nous distinguons plusieurs actions. D'abord, l'animation qui correspond à l'état `_down` est lancée :

```
evt.target.gotoAndPlay("_down");
```

Ensuite, nous créons une propriété `etatVisit`, pour chaque bouton activé, que nous passons sur `true`. Cette propriété nous sert à vérifier si l'objet a été cliqué ou non. Selon sa valeur, nous définissons plus loin le type d'état à afficher en sortie ("`_visited`" ou "`_sortie`").

Nous ajoutons ensuite des conditions qui permettent de lancer telle ou telle action, selon le bouton cliqué. Nous vérifions, par exemple, que le bouton cliqué se nomme `bouton1_mc`. Dans ce cas, un texte en rapport s'affiche dans la fenêtre de sortie.

C'est la définition de différentes structures conditionnelles qui nous épargne d'avoir à créer un nouveau gestionnaire d'événements propre à chaque bouton. L'ensemble des actions du menu est centralisé dans chacune de ces conditions.

Ensuite, un gestionnaire affiche les actions lorsque le pointeur sort des boutons (`MOUSE_OUT`) :

```
//out
menu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    if (evt.target.etatVisit==true) {
        evt.target.gotoAndPlay("_visited");
    } else {
        evt.target.gotoAndPlay("_out");
    }
}
```

Dans cette fonction, nous définissons deux actions. La première active l'interpolation située à l'étiquette `_visited` si, et uniquement si, la valeur de la propriété `etatVisit` du bouton cliqué est passée sur `true`, c'est-à-dire, si le bouton, ou la rubrique associée à ce bouton, a bien été visitée.

Si le bouton pour lequel le pointeur sort n'est pas le bouton cliqué, la seconde action appelle l'interpolation de sortie simple (`_out`).

La structure *Switch... case*

Vous trouverez également, dans les exercices du livre, le fichier "ch13_systemesNavigation_1b fla". Ce document propose une version alternative de la structure conditionnelle. La fonction intitulée `down` emploie ici une instruction `switch... case`, au lieu de `if` :

```
//down
menu_mc.addEventListener(MouseEvent.CLICK,down);
function down (evt:MouseEvent) {
    evt.target.gotoAndPlay("_down");
    evt.target.etatVisit=true;
    //
    switch (evt.target.name) {
        case "bouton1_mc":
            trace("action 1");
            break;
        case "bouton2_mc":
            trace("action 2");
            break;
        case "bouton3_mc":
            trace("action 3");
            break;
        default:
            trace("ni 0, 1, ou 2");
    }
}
```

Le principe de l'instruction `switch` est similaire à une structure `if`, à la différence que nous vérifions une seule fois l'objet de la condition (`switch (evt.target.name)`). Selon la valeur identifiée après l'attribut `case`, nous spécifions ensuite les actions à exécuter. Les actions sont placées après les deux points et se terminent toujours par l'instruction `break`. Une instruction `default` permet d'exécuter une autre action lorsque la condition n'est vérifiée par aucun attribut `case`.

Cette structure est particulièrement adaptée pour les systèmes dont la condition à vérifier est toujours du même type, comme vérifier une valeur portée par une variable, et y associer un comportement en fonction du résultat obtenu.

Les boutons activés

Un troisième document, nommé "ch13_systemesNavigation_1c.fl", propose un système de navigation pour lequel chaque lien affiche l'état actif lorsque le bouton vient d'être enfoncé. Mais cet état est désormais restauré dès qu'un autre bouton reprend le focus.

Dans ce document, nous pouvons lire le code suivant :

```
//----- initialisation
var actif:*=null;

//----- actions menu
//down
menu_mc.addEventListener(MouseEvent.CLICK,down);
function down(evt:MouseEvent) {
    if (evt.target!=actif) {
        //up
        if (actif!=null) {
            actif.gotoAndPlay("_up");
        }
        //visited
        actif=evt.target;
        actif.gotoAndPlay("_visited");
    }
    switch (evt.target.name) {
        case "bouton1_mc":
            trace("action 1");
            break;
        case "bouton2_mc":
            trace("action 2");
            break;
        case "bouton3_mc":
            trace("action 3");
            break;
        default:
            trace("ni 0, 1, ou 2");
    }
}
//over
menu_mc.addEventListener(MouseEvent.CLICK,over);
function over(evt:MouseEvent) {
    if (evt.target!=actif) {
        evt.target.gotoAndPlay("_over");
    }
}
//out
menu_mc.addEventListener(MouseEvent.CLICK,out);
function out(evt:MouseEvent) {
    if (evt.target!=actif) {
        evt.target.gotoAndPlay("_out");
    }
}
```

Dans ce programme, nous initialisons d'abord une variable "interrupteur" que nous avons nommée actif :

```
//----- initialisation  
var actif:*=null;
```

Le type étoile (*) désigne tout type, sans en privilégier un en particulier. La valeur null assure qu'elle ne permettra aucune action tant que nous ne l'aurons pas modifiée.

La fonction down commence par modifier cette valeur lorsqu'un bouton est enfoncé (actif=evt.target). Les autres fonctions ainsi que les structures conditionnelles de celle-ci, vérifient alors que le bouton cliqué n'est pas celui préalablement activé. Selon la valeur détectée, elles lancent ou non les interpolations d'entrée (_over) ou de sortie (_out) des boutons du menu.



Menus avec des symboles boutons. Il est possible de réaliser dynamiquement un menu à partir de symboles boutons et de cibler les contenus qui y sont distribués. Nous n'abordons pas cet aspect ici. Ces techniques sont décrites en détail au Chapitre 7 de l'ouvrage de Thibault Imbert, *Pratique d'ActionScript 3*, aux éditions Pearson.

À retenir

- Les boutons réalisés en MovieClip offrent plus de souplesse de manipulation et de valeur ajoutée créative, qu'un bouton classique de la classe Button. Il est notamment possible d'y introduire de nombreuses animations et de les contrôler facilement par ActionScript.
- Pour créer un état visité sur un bouton cliqué, nous devons ajouter une condition qui vérifie si le bouton activé est celui enregistré. Dans ce cas, nous jouons l'animation en rapport. Le cas échéant, nous spécifions une autre interpolation.
- En ActionScript 3, grâce à la propriété target, il est facile de centraliser les actions de tout un menu au travers d'un seul gestionnaire d'événements. Pour distinguer néanmoins les actions de chaque bouton, nous utilisons des structures conditionnelles.
- Une structure conditionnelle de type Switch est plus adaptée qu'une structure if, lorsque la condition à vérifier est toujours la même.

Boutons MovieClip animés et interfaçables

La création de boutons classiques consiste à limiter la surface d'affichage à leur libellé. Les boutons de type interfaçables sont des MovieClip qui exécutent une animation et transforment la surface du bouton en fenêtre de contenu. Les éléments distribués dans cette fenêtre deviennent à leurs tours interactifs, mais ne doivent pas entrer en conflit avec les actions placées justement sur ce conteneur (voir Figure 13.5).

La difficulté de cet exemple repose sur la capacité à rendre les actions des éléments ajoutés, dans le conteneur MovieClip, indépendantes des actions du conteneur lui-même. Si le bouton qui déploie cette fenêtre possède ses propres actions d'ouverture et de fermeture.

Figure 13.5

Bouton interfacé.



Il devient effectivement plus compliqué d'ajouter, dans un objet déjà interactif, d'autres objets également interactifs.

Afin de permettre de placer des symboles cliquables à l'intérieur d'autres symboles déjà cliquables, nous devons considérer que :

- Les objets placés au-dessus des autres sont toujours prioritaires sur les actions à exécuter.
- Les objets doivent pouvoir être identifiés au moment précis où des actions leur sont attribuées. Ou bien, il n'est pas possible de cibler un objet qui n'apparaît pas dans la scène au moment où celui-ci est invoqué.
- En ActionScript 3, l'ensemble des objets enfants d'un conteneur remonte au conteneur principal les interactions éventuellement interceptées.

Pour ajouter des objets interactifs dans un MovieClip, lui-même interactif, nous procédons de l'une des deux manières suivantes. Soit nous plaçons tous les objets interactifs sur la première image du scénario de leur conteneur, quitte à les rendre invisibles (propriété `visible=false`) afin qu'ils n'apparaissent pas au lancement de l'application et, qu'ils ne soient pas cliquables. Soit nous les isolons sur une image du scénario du conteneur, là où effectivement ils doivent apparaître, mais nous sommes alors contraints de placer les actions de ces objets à ce même emplacement, dans le scénario.

Dans cette section, nous abordons les deux solutions.



Différence entre les propriétés alpha et visible.

La propriété `alpha` (`nomDuSymbole_mc.alpha=0`) rend les objets transparents mais toujours actifs. La propriété `visible` (`nomDuSymbole_mc.visible=false`) les rend invisibles et inactifs. L'alpha se définit par une valeur décimale comprise entre 0 et 1. La visibilité se définit par une valeur booléenne (`true` ou `false`).

Méthode avec les actions dans le scénario du lien

Nous présentons d'abord la solution qui consiste à placer les actions des liens imbriqués, dans les liens eux-mêmes. Cette technique est la plus simple à appréhender. Elle permet de placer les objets dans le scénario uniquement lorsqu'ils doivent être actifs. Elle convient parfaitement pour des actions localisées, comme la lecture du scénario à partir d'un objet ajouté localement, sur une image-clé isolée, par exemple.



Exemples > ch13_systemesNavigation_2 fla

Exemples > ch13_systemesNavigation_2b fla

Dans le document ch13_systemesNavigation_2 fla, nous pouvons voir une interface où des liens, en forme de poids, sont positionnés au-dessus de la partie gauche du décor. Un MovieClip canalise ces objets et porte le nom d'occurrence menu_mc (voir Figure 13.6).

Figure 13.6

Aperçu de la scène principale.



À l'intérieur de ce menu, trois symboles de type MovieClip, respectivement nommés bouton1_mc, bouton2_mc et bouton3_mc sont répartis distinctement vers les calques (voir Figure 13.7).

Figure 13.7

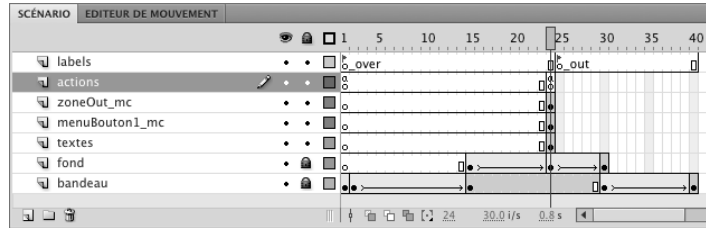
Scénario à l'intérieur du menu.



À l'intérieur du symbole bouton1_mc, nous accédons maintenant à une animation où la surface du lien s'étend jusqu'à dessiner une fenêtre. Cette animation prend fin à l'image 24 où la fenêtre est totalement déployée. Le reste de l'animation représente la fermeture de cette fenêtre. La lecture de cette animation est contrôlée par des actions. Un stop, placé à l'image 1 et 24, interrompt la lecture. C'est uniquement, lorsque la tête de lecture atteint l'image 24, que de nouveaux boutons apparaissent à l'écran (voir Figure 13.8).

Figure 13.8

Scénario à l'intérieur du bouton 1.



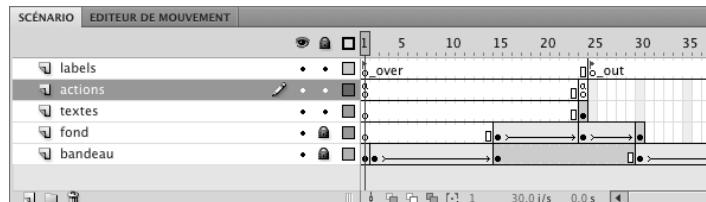
Dans la fenêtre de scénario du bouton 1, un autre objet apparaît également à l'image 24, un movieClip transparent nommé zoneOut_mc et de propriété alpha à 0 %. C'est lui qui permet d'activer la fermeture de la fenêtre, lorsque le pointeur le dépasse.

Afin d'activer l'ouverture animée de cette fenêtre, nous avons placé des actions sur la scène principale, en direction de l'objet bouton1_mc lui-même. Pour que les liens situés localement à l'image 24 du symbole bouton1_mc soient actifs, nous avons placé d'autres actions directement dans ce scénario, à l'image 24.

Les autres MovieClip boutons de ce document ne contiennent pas de liens isolés. Ils ont été mis en forme uniquement pour illustrer le mécanisme, dans un système plus global, avec plusieurs entrées de menu. Leur scénario ne possède pas les calques menuBouton1_mc ni zoneOut_mc (voir Figure 13.9). Mais le mécanisme adopté pour le premier bouton pourrait y être reproduit.

Figure 13.9

Scénario à l'intérieur des boutons 2 et 3.



Les actions du calque de la scène principale sont les suivantes :

```
//----- initialisation
var boutonActif:String;

//----- actions
//over
menu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    if (evt.target.currentFrame==1) {
        evt.target.gotoAndPlay("_over");
    }
}
```

```

    }
    //down
    menu_mc.addEventListener(MouseEvent.CLICK,down);
    function down (evt:MouseEvent) {
        boutonActif=evt.target.name;
        //
        if (evt.target.name=="bouton1_mc") {
            trace("action 1");
        }
        if (evt.target.name=="bouton2_mc") {
            trace("action 2");
        }
        if (evt.target.name=="bouton3_mc") {
            trace("action 3");
        }
    }
    //out
    menu_mc.addEventListener(MouseEvent.CLICK,out);
    function out (evt:MouseEvent) {
        if (evt.target.name!="bouton1_mc") {
            if (evt.target.currentFrame==24) {
                evt.target.gotoAndPlay("_out");
            }
        }
    }
}

```

Les actions localisées dans le symbole bouton1_mc, à l'image 24, sont les suivantes :

```

stop();
//
menuBouton1_mc.addEventListener(MouseEvent.CLICK,actionsMenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
    if (evt.target.name=="lien1_mc") {
        trace("action 1 du menu bouton 1");
    }
    if (evt.target.name=="lien2_mc") {
        trace("action 2 du menu bouton 1");
    }
    if (evt.target.name=="lien3_mc") {
        trace("action 3 du menu bouton 1");
    }
}
//
zoneOut_mc.addEventListener(MouseEvent.CLICK,sortirBouton1);
function sortirBouton1 (evt:MouseEvent) {
    play();
}

```

Dans la fenêtre d'actions de la scène principale, nous associons une fonction sur le conteneur menu_mc qui accueille les trois boutons :

```

//----- actions
//over
menu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    if (evt.target.currentFrame==1) {
        evt.target.gotoAndPlay("_over");
    }
}

```


Dans cette fonction, nous activons la lecture du scénario du bouton cliqué, uniquement si l'image courante du bouton cliqué est l'image 1. Ceci évite que l'animation soit relancée intempestivement lorsque l'utilisateur navigue au sein de la fenêtre, si elle est déployée.

Plus bas, dans la fonction associée à l'événement `MOUSE_DOWN`, nous reprenons le même type d'actions que celles présentées dans la section précédente.

En revanche, dans le gestionnaire `MOUSE_OUT`, nous précisons que l'action de restauration de l'état du bouton ne peut avoir lieu que si le nom du bouton actif est différent de celui qui contient d'autres liens. Autrement dit, l'action de restauration n'apparaît que pour les liens simples. L'action de fermeture de la fenêtre, dans notre exemple, est contrôlée directement à l'image 24, dans le scénario du MovieClip bouton :

```
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
    if (evt.target.name!="bouton1_mc") {
        if (evt.target.currentFrame==24) {
            evt.target.gotoAndPlay("_out");
        }
    }
}
```

Dans le scénario du MovieClip, à l'image 24, nous affectons d'abord un écouteur sur l'enveloppe `menuBouton1_mc` qui rassemble les trois sous-entrées locales. Pour chaque lien de ce conteneur, nous définissons une action trace spécifique :

```
menuBouton1_mc.addEventListener(MouseEvent.CLICK,actionsMenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
    if (evt.target.name=="lien1_mc") {
        trace("action 1 du menu bouton 1");
    }
    if (evt.target.name=="lien2_mc") {
        trace("action 2 du menu bouton 1");
    }
    if (evt.target.name=="lien3_mc") {
        trace("action 3 du menu bouton 1");
    }
}
```

Il n'y a pas de contradiction avec le conteneur principal, car ces objets sont situés au premier plan, et le gestionnaire cible bien un conteneur différent du conteneur principal.

Plus bas, une autre action est ajoutée et contrôle la fermeture de la fenêtre :

```
//
zoneOut_mc.addEventListener(MouseEvent.MOUSE_OUT,sortirBouton1);
function sortirBouton1 (evt:MouseEvent) {
    play();
}
```

L'action cible tout simplement la forme transparente située au pourtour de la fenêtre. Lorsque cette zone apparaît dans la scène, le pointeur est déjà sur la partie centrale. L'événement `MOUSE_OUT` permet donc d'activer la fermeture animée de la fenêtre, uniquement si l'utilisateur sort de la zone transparente, par l'extérieur.

Des instructions de neutralisation de MovieClip et de boutons sont proposées en fin de chapitre. Ils offrent une autre alternative à la gestion de contenus imbriqués.

Dans le document "ch13_systemesNavigation_2b.fla", une déclinaison de cet exemple est proposée, à partir de l'instruction switch. Ce qui donne, pour la fonction down de la scène principale :

```
//down
function down (evt:MouseEvent) {
    boutonActif=evt.target.name;
    //
    switch (evt.target.name) {
        case "bouton1_mc":
            trace("action 1");
            break;
        case "bouton2_mc":
            trace("action 2");
            break;
        case "bouton3_mc":
            trace("action 3");
            break;
        default:
            trace("ni 0, 1, ou 2");
    }
}
```

Pour la fonction placée à l'intérieur du symbole :

```
menuBouton1_mc.addEventListener(MouseEvent.CLICK,actionsMenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
    switch (evt.target.name) {
        case "lien1_mc":
            trace("action 1 du menu bouton 1");
            break;
        case "lien2_mc":
            trace("action 2 du menu bouton 1");
            break;
        case "lien3_mc":
            trace("action 3 du menu bouton 1");
            break;
        default:
            trace("aucun");
    }
}
```

Méthode avec les actions sur la scène principale

Une déclinaison du précédent document est proposée dans le fichier "ch13_systemesNavigation_2c.fla". Ce document prend en charge la gestion des liens imbriqués en plaçant le code directement sur la scène principale, ce qui rend plus facile la liaison des données entre fonctions et simplifie, entre autres, la récupération de valeurs, l'activation d'autres fonctions, la centralisation du code et sa maintenance.

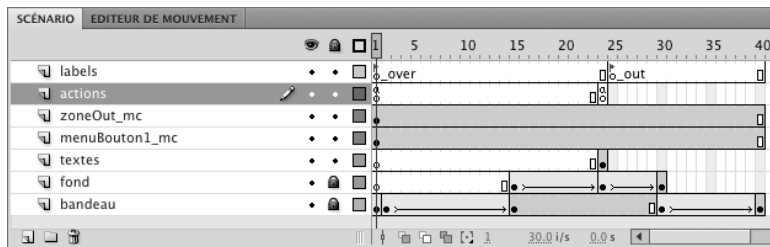


Exemples > ch13_systemesNavigation_2c.fl

Dans ce document, nous observons que l'intérieur du MovieClip bouton1_mc affiche déjà les objets zoneOut_mc et menuBouton1_mc, dès la première image (voir Figure 13.10).

Figure 13.10

Scénario à l'intérieur du bouton 1.



D'autre part, aucune action spécifique, en dehors des stops, n'apparaît non plus à l'image 24. Toutes les actions sont centralisées à l'image 1 du scénario de la scène principale.

Dans le code de la scène principale, nous pouvons lire ceci :

```
//----- initialisation
var boutonActif:String;

//----- actions
//over
menu_mc.addEventListener(MouseEvent.CLICK,over);
function over (evt:MouseEvent) {
    if (evt.target.currentFrame==1) {
        evt.target.gotoAndPlay("_over");
    }
}
//down
menu_mc.addEventListener(MouseEvent.CLICK,down);
function down (evt:MouseEvent) {
    boutonActif=evt.target.name;
    //
    if (evt.target.name=="bouton1_mc") {
        trace("action 1");
    }
    if (evt.target.name=="bouton2_mc") {
        trace("action 2");
    }
    if (evt.target.name=="bouton3_mc") {
        trace("action 3");
    }
}
//out
menu_mc.addEventListener(MouseEvent.CLICK,out);
function out (evt:MouseEvent) {
    if (evt.target.name!="bouton1_mc") {
        if (evt.target.currentFrame==24) {
            evt.target.gotoAndPlay("_out");
        }
    }
}
```

```

    }
}

// liens 1, 2 et 3, dans bouton 1
menu_mc.bouton1_mc.menuBouton1_mc.addEventListener(MouseEvent.CLICK,actions-
➡ MenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
    if (evt.target.name=="lien1_mc") {
        trace("action 1 du menu bouton 1");
    }
    if (evt.target.name=="lien2_mc") {
        trace("action 2 du menu bouton 1");
    }
    if (evt.target.name=="lien3_mc") {
        trace("action 3 du menu bouton 1");
    }
}
menu_mc.bouton1_mc.menuBouton1_mc.visible=false;

// sortie bouton 1
menu_mc.bouton1_mc.zoneOut_mc.addEventListener(MouseEvent.MOUSE_OUT,sortirBouton1);
function sortirBouton1 (evt:MouseEvent) {
    menu_mc.bouton1_mc.play();
}
menu_mc.bouton1_mc.zoneOut_mc.visible=false;

// gestion de l'affichage des liens dans le bouton 1
addEventListener(Event.ENTER_FRAME,afficherLiens);
function afficherLiens (evt:Event) {
    if (menu_mc.bouton1_mc.currentFrame==24) {
        menu_mc.bouton1_mc.menuBouton1_mc.visible=true;
        menu_mc.bouton1_mc.zoneOut_mc.visible=true;
    } else {
        menu_mc.bouton1_mc.menuBouton1_mc.visible=false;
        menu_mc.bouton1_mc.zoneOut_mc.visible=false;
    }
}
}

```

Dans ce document, nous avons déplacé les actions de l'image 24, du symbole MovieClip bouton1_mc, à l'intérieur de la fenêtre actions de la scène principale. Afin que ces actions s'exécutent correctement, nous avons placé les objets ciblés sur la première image du scénario bouton1. Ainsi, ils peuvent être identifiés par la tête de lecture, au moment où les actions de la scène principale sont interprétées. Mais, pour que ces objets n'apparaissent pas au démarrage, nous avons ajouté des contrôles de visibilité, en fonction de l'image active de ce scénario.

Le même document, nommé "ch13_systemesNavigation_2d fla", décliné avec des instructions switch, est également disponible dans les fichiers des exemples du livre.

À retenir

- Des liens peuvent être placés localement dans un objet déjà interactif. Pour cela, ils doivent être placés au premier plan et leurs actions doivent être situées localement au niveau du lien.
- Une approche alternative existe. Il est possible de placer les actions sur la scène principale, mais avec des conditions qui neutralisent d'abord l'affichage de l'objet par défaut.

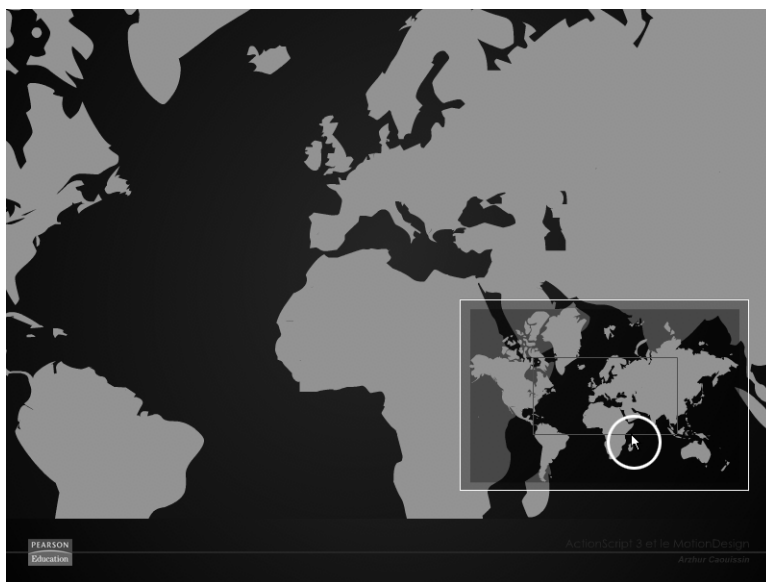
Console de navigation en miniature

Si vous utilisez la fenêtre de navigation de Photoshop ou d'Illustrator, vous êtes sûrement familier avec la navigation dans une fenêtre réduite. Une fenêtre de navigation réduite reprend, en miniature, un aperçu de l'ensemble de la scène et permet à l'utilisateur de s'y mouvoir en Y déplaçant un simple rectangle. Ce rectangle matérialise la zone visible à l'écran.

Dans cette section, nous allons voir comment mettre en place ce dispositif de navigation. Nous utilisons pour cela une carte du monde, agrandie à l'échelle de 200 % par rapport à sa taille initiale. Cette carte déborde donc largement de la zone visible par l'utilisateur. C'est en déplaçant le rectangle rouge de la fenêtre de navigation, que l'on fera bouger le contenu, pour accéder aux informations que celui-ci propose (voir Figure 13.11).

Figure 13.11

Aperçu du document publié.



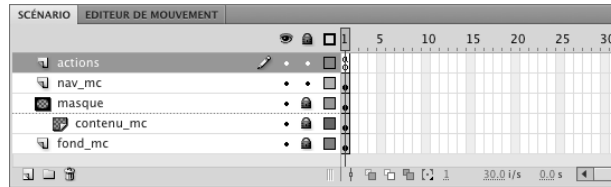
Exemples > ch13_systemesNavigation_3 fla

Dans le document "ch13_systemesNavigation_3 fla", sur la scène principale (voir Figure 13.12), au-dessus du calque fond_mc, nous visualisons le calque contenu_mc

masqué par un rectangle de dimensions équivalentes à celles de la scène, moins la barre d'informations.

Figure 13.12

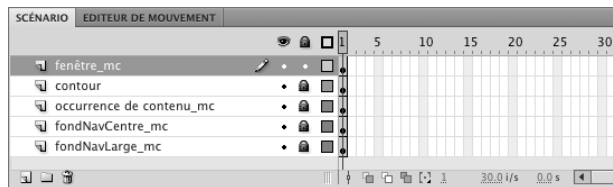
Fenêtre de scénario de la scène principale.



À l'intérieur du calque `nav_mc`, figurent plusieurs symboles, dont le rectangle rouge que nous utilisons comme outil de déplacement et qui porte le nom d'occurrence `fenetre_mc` (voir Figure 13.13). En dessous, une copie de l'occurrence `contenu_mc` est affichée en proportions réduites, elle illustre l'ensemble du contenu disponible à la navigation, et situe la position courante du rectangle par rapport au contenu affiché dans cette zone. En arrière plan de ces calques, le symbole `fondNavCentre_mc` représente la zone limite de déplacement dont nous reprenons les coordonnées dans le code. L'objet `fondNavLarge_mc` et le calque contour servent uniquement la décoration.

Figure 13.13

Fenêtre de scénario du symbole `nav_mc`.



Dans la fenêtre Actions de la scène principale, nous pouvons lire le code suivant :

```
//----- initialisation
var origineX:Number=nav_mc.fondNavCentre_mc.x;
var origineY:Number=nav_mc.fondNavCentre_mc.y;
var largeurRect:Number=(nav_mc.fondNavCentre_mc.width)-
➤ (nav_mc.fenetre_mc.width);
var hauteurRect:Number=(nav_mc.fondNavCentre_mc.height)-
➤ (nav_mc.fenetre_mc.height);
var zoneDeplacement:Rectangle=new Rectangle(origineX, origineY, largeurRect,
➤ hauteurRect);
//
var echelleMouvement:Number=(contenu_mc.width/nav_mc.fondNavCentre_mc.width);

//----- actions
// déplacement du rectangle
nav_mc.fenetre_mc.addEventListener(MouseEvent.CLICK,deplacerRectangle);
function deplacerRectangle (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME, deplacerContenu);
    nav_mc.fenetre_mc.startDrag(false,zoneDeplacement);
}
addEventListener(MouseEvent.CLICK,relacherRectangle);
function relacherRectangle (evt:MouseEvent) {
```

```

        removeEventListener(Event.ENTER_FRAME, deplacerContenu);
        stopDrag();
    }

    // déplacement du contenu
    function deplacerContenu (evt:Event) {
        contenu_mc.x=nav_mc.fenêtre_mc.x*-echelleMouvement;
        contenu_mc.y=nav_mc.fenêtre_mc.y*-echelleMouvement;
    }

```

Le programme se déroule en trois étapes. Nous définissons d'abord la zone de déplacement. Nous activons ensuite le mouvement. Puis nous lions le contenu de la scène à l'outil déployé et mobile.

Tout d'abord, dans la partie initialisation, nous spécifions, au travers des quatre premières variables, les limites de la zone de déplacement :

```

var origineX:Number=nav_mc.fondNavCentre_mc.x;
var origineY:Number=nav_mc.fondNavCentre_mc.y;
var largeurRect:Number=(nav_mc.fondNavCentre_mc.width)-
➤ (nav_mc.fenêtre_mc.width);
var hauteurRect:Number=(nav_mc.fondNavCentre_mc.height)-
➤ (nav_mc.fenêtre_mc.height);
var zoneDeplacement:Rectangle=new Rectangle(origineX, origineY, largeurRect,
➤ hauteurRect);

```

Nous reprenons la position courante et les dimensions de la zone de déplacement contenue dans le symbole `nav_mc`, qui porte le nom d'occurrence `fondNavCentre_mc`. Mais pour définir la largeur utile de ce déplacement, nous devons également soustraire la largeur et la hauteur du rectangle mobile, comme nous l'avions fait pour l'ascenseur au Chapitre 2, avec les interpolations `TweenMax`.

Plus bas, nous définissons une variable pour déterminer le coefficient de vitesse du mouvement. Il s'agit du rapport d'échelle entre la taille du contenu de la scène principale et les dimensions de la zone de déplacement :

```

var echelleMouvement:Number=(contenu_mc.width/nav_mc.fondNavCentre_mc.width);

```

Dans la deuxième partie, nous activons en premier la fonction qui recalcule la position des objets (`deplacerContenu`). Puis, à la ligne, nous activons le contrôle du mouvement de la forme rectangulaire rouge et introduisons, en paramètre de la méthode `startDrag`, la référence au rectangle préalablement désigné :

```

//----- actions
// déplacement du rectangle
nav_mc.fenêtre_mc.addEventListener(MouseEvent.MOUSE_DOWN,deplacerRectangle);
function deplacerRectangle (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME, deplacerContenu);
    nav_mc.fenêtre_mc.startDrag(false,zoneDeplacement);
}
addEventListener(MouseEvent.MOUSE_UP,relacherRectangle);
function relacherRectangle (evt:MouseEvent) {
    removeEventListener(Event.ENTER_FRAME, deplacerContenu);
    stopDrag();
}

```

Cette forme est entièrement cliquable car nous y avons introduit un symbole qui couvre toute la surface mais avec un `alpha` à 0 %. Le paramètre `false` indique que l'objet ne se repositionne pas en fonction du clic souris, à l'inverse, l'usage de `true` aurait centré le rectangle en fonction de ce clic.

La fonction `relacherRectangle` interrompt le déplacement du symbole et la fonction `deplacerContenu`, gérée par un `ENTER_FRAME`.

Enfin, nous ajoutons un écouteur qui exécute le repositionnement en reprenant le coefficient calculé en amont, et repositionne le symbole `contenu_mc`, en X et en Y, en fonction de la position du rectangle rouge :

```
// déplacement du contenu
addEventListener(Event.ENTER_FRAME, deplacerContenu);
function deplacerContenu (evt:Event) {
    contenu_mc.x=nav_mc.fenêtre_mc.x*-echelleMouvement;
    contenu_mc.y=nav_mc.fenêtre_mc.y*-echelleMouvement;
}
```

À retenir

- La partie active d'un bouton ou de tout type de symbole doit toujours être matérialisée. Pour déplacer une forme qui n'affiche pas de fond, nous plaçons, à l'intérieur du symbole utilisé pour le déplacement, une forme pleine mais transparente.
- Pour déterminer le rapport d'échelle des dimensions entre un contenu et la scène courante, il suffit de diviser les dimensions de la première par la seconde.

Menu déroulant à repositionnement automatique

La création d'un menu déroulant ne pose pas de difficulté particulière dès lors que l'on sait utiliser les masques, le scénario, quelques transitions animées et les conditions en ActionScript. Mais, si nous voulons que les menus s'affichent en colonne, par exemple, les uns au-dessus des autres, le mécanisme peut se compliquer. En effet, pour ouvrir un menu et afficher ses sous-entrées, nous devons alors chasser les menus du dessous, encore refermés, à mesure que les sous-entrées du menu précédent descendent. Et inversement à la fermeture.

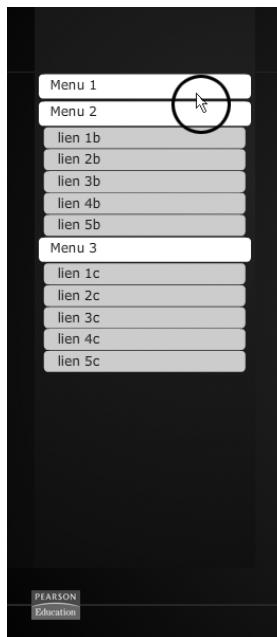
Pour réaliser ce dispositif, nous créons des sous-menus directement à l'intérieur des MovieClip boutons qui représentent les entrées principales de chaque menu. Puis, nous utilisons des transitions de type `TweenMax` pour activer le défilement des sous-menus, sur clic de l'utilisateur.

Nous gérons aussi le repositionnement des entrées principales de menu, avec un gestionnaire de type `ENTER_FRAME`. C'est dans ce gestionnaire que nous indiquons, à chaque entrée principale, de se positionner en fonction de la position Y du dernier élément du menu précédent. Ainsi, quelle que soit la position des sous-entrées de chaque menu, les entrées principales et leurs sous-menus respectifs, enfants, suivent les mouvements de chaque sous-menu.

Dans cette section, nous utilisons un menu à trois entrées, qui affichent, chacune, cinq sous-entrées. Les symboles utilisés sont tous différents. Vous pouvez donc rapidement personnaliser le fichier pour vous l'approprier et y placer de vraies fonctionnalités (voir Figure 13.14).

Figure 13.14

Aperçu du document publié.



Exemples > ch13_systemesNavigation_4.fla

Dans la scène principale du document "ch13_systemesNavigation_4.fla", nous pouvons voir un symbole menu_mc qui contient trois entrées principales de menu, respectivement nommées entree1_mc, entree2_mc et entree3_mc.

Dans le scénario de chaque bouton MovieClip d'entrée, nous remarquons la présence d'un masque qui rend invisibles les sous-entrées appelées par les scripts. Si nous déverrouillons les calques, nous constatons que ces sous-entrées sont positionnées, au pixel près, au-dessus de la zone d'affichage, matérialisée par le masque (voir Figures 13.15 et 13.16). Dans les scripts, nous déterminerons le positionnement des éléments en fonction de leur position courante. Il est donc important que les éléments ne soient pas placés n'importe où.

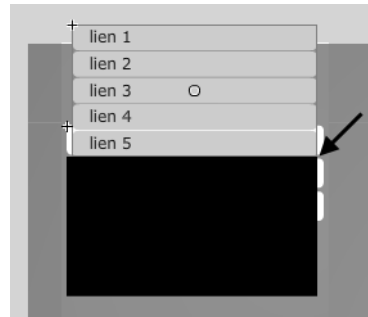
Figure 13.15

Aperçu du scénario de la scène d'un sous-menu

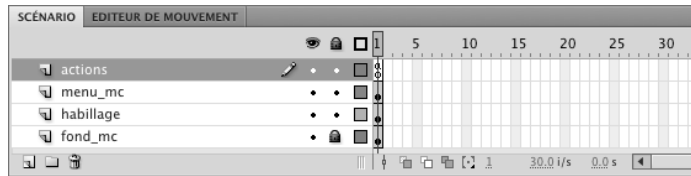


Figure 13.16

Aperçu du sous-menu et de son masque.

**Figure 13.17**

Aperçu du scénario de la scène principale.



Dans le scénario de la scène principale (voir Figure 13.17), au-dessus des calques de décoration et du menu, nous accédons aux actions qui affichent le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

var initEntree1Y:Number=menu_mc.entree1_mc.y;
var initEntree2Y:Number=menu_mc.entree2_mc.y;
var initEntree3Y:Number=menu_mc.entree3_mc.y;

var initSousMenu1Y:Number=menu_mc.entree1_mc.sousMenu_mc.y;
var initSousMenu2Y:Number=menu_mc.entree2_mc.sousMenu_mc.y;
var initSousMenu3Y:Number=menu_mc.entree3_mc.sousMenu_mc.y;

//----- actions
// activation des sous-menus
menu_mc.addEventListener(MouseEvent.CLICK,ouvrirMenu);
function ouvrirMenu(evt:MouseEvent) {
    if (evt.target.name=="entree1_mc") {
        if (menu_mc.entree1_mc.sousMenu_mc.y<=initSousMenu1Y) {
            TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
                y:initSousMenu1Y + menu_mc.entree1_mc.sousMenu_mc.height,
                ease:Strong.easeInOut,
                onStartListener : placeEcouteur,
                onCompleteListener :retireEcouteur,
                overwrite:3
            });
        } else {
```

```

        TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
        });
    }
}
if (evt.target.name=="entree2_mc") {
    if (menu_mc.entree2_mc.sousMenu_mc.y<=initSousMenu2Y) {
        TweenMax.to(menu_mc.entree2_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y + menu_mc.entree2_mc.sousMenu_mc.height,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
        });
    } else {
        TweenMax.to(menu_mc.entree2_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
        });
    }
}
if (evt.target.name=="entree3_mc") {
    if (menu_mc.entree3_mc.sousMenu_mc.y<=initSousMenu3Y) {
        TweenMax.to(menu_mc.entree3_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y + menu_mc.entree3_mc.sousMenu_mc.height,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
        });
    } else {
        TweenMax.to(menu_mc.entree3_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
        });
    }
}
}
trace(evt.target.name);
}

// suivi des entrées
function placeEcouteur(Evt:TweenEvent){
    addEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
function retireEcouteur(Evt:TweenEvent){
    removeEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}

```

```
function remplacerEntreesMenu(evt:Event) {
    trace("fonction active")
    menu_mc.entree2_mc.y=initEntree2Y+(menu_mc.entree1_mc.sousMenu_mc.
    ➤ y-initSousMenu1Y);
    menu_mc.entree3_mc.y=initEntree3Y+(menu_mc.entree1_mc.sousMenu_mc.
    ➤ y-initSousMenu1Y)+(menu_mc.entree2_mc.sousMenu_mc.y-initSousMenu2Y);
}
```

Notre code se définit en trois parties. La première importe les classes et enregistre la position courante des éléments. La deuxième active le déroulement des sous-menus. La troisième et dernière partie repositionne les entrées en fonction de la position courante des sous-menus.

Dans la première partie, l'initialisation, après l'importation des classes, nous stockons les valeurs de position en Y, de chaque sous-menu et de chaque entrée de menu :

```
var initEntree1Y:Number=menu_mc.entree1_mc.y;
var initEntree2Y:Number=menu_mc.entree2_mc.y;
var initEntree3Y:Number=menu_mc.entree3_mc.y;

var initSousMenu1Y:Number=menu_mc.entree1_mc.sousMenu_mc.y;
var initSousMenu2Y:Number=menu_mc.entree2_mc.sousMenu_mc.y;
var initSousMenu3Y:Number=menu_mc.entree3_mc.sousMenu_mc.y;
```

Dans les actions, lorsque l'utilisateur active une des entrées du menu (evt.target), nous ajoutons des actions selon le nom de l'entrée activée :

```
//----- actions
// activation des sous-menus
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,ouvrirMenu);
function ouvrirMenu(evt:MouseEvent) {
    if (evt.target.name=="entree1_mc") {
        if (menu_mc.entree1_mc.sousMenu_mc.y<=initSousMenu1Y) {
            TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
                y:initSousMenu1Y + menu_mc.entree1_mc.sousMenu_mc.height,
                ease:Strong.easeInOut,
                onStartListener : placeEcouteur,
                onCompleteListener :retireEcouteur,
                overwrite:3
            });
        } else {
            TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
                y:initSousMenu1Y,
                ease:Strong.easeInOut,
                onStartListener : placeEcouteur,
                onCompleteListener :retireEcouteur,
                overwrite:3
            });
        }
    }
}
// déclinaison des instructions pour les entrées de menu 2 et 3
trace(evt.target.name);
}
```

À l'intérieur de chaque condition, une autre condition vérifie si la position courante du sous-menu concerné est en position affichée ou repliée. La position repliée est définie en reprenant une des valeurs enregistrées en amont. Si la valeur enregistrée est identique à la position actuelle, alors, le sous-menu n'a pas bougé. Il est donc ramassé. Ceci s'inverse dès que le sous-menu est activé, celui-ci est déplacé et sa position ne répond plus à la condition qui alors exécute une autre instruction.

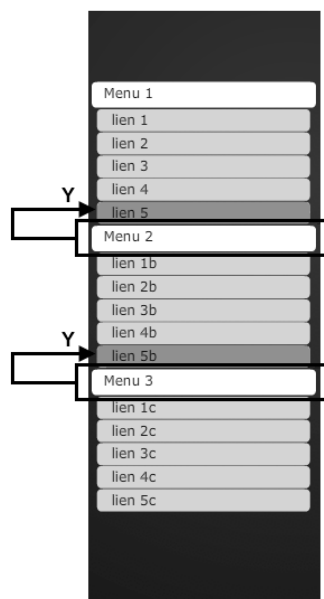
La position des sous-menus, lorsqu'ils sont affichés, augmente de l'équivalent de leur hauteur respective (d'où la nécessité de les placer précisément à la limite de la zone d'affichage, dans chaque entrée de menu) :

```
TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
    y: initSousMenu1Y + menu_mc.entree1_mc.sousMenu_mc.height,
    ease: Strong.easeInOut,
    onStartListener : placeEcouteur,
    onCompleteListener : retireEcouteur,
    overwrite: 3
});
```

Si la condition est vérifiée, si le menu est fermé donc, une transition de type TweenMax le déplace de l'équivalent de sa hauteur à laquelle nous ajoutons (par sécurité), sa position enregistrée. Sinon, il le ramène uniquement à sa position préalablement enregistrée (initSousMenu1Y), donc, en position replié (voir Figure 13.18).

Figure 13.18

Mécanisme de repositionnement des entrées principales du menu.



Au sein des interpolations, d'autres propriétés sont utilisées. Chacune se termine en invoquant directement, grâce aux propriétés d'écouteur `onStartListener` et `onCompleteListener`, abordées au Chapitre 2, deux autres fonctions. La première (`placeEcouteur`) active le repositionnement continu des objets au démarrage de l'interpolation. La seconde (`retireEcouteur`) interrompt cette fonction une fois l'interpolation achevée :

```
// suivi des entrées
function placeEcouteur(Evt:TweenEvent){
    addEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
function retireEcouteur(Evt:TweenEvent){
    removeEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
```

Il serait dommage en effet de conserver active la fonction `replacerEntreeMenu` et de continuer de calculer le repositionnement même lorsqu'aucune entrée du menu n'est cliquée. Cette fonction qui est associée à l'événement `ENTER_FRAME` sollicite d'importantes ressources. En désactivant cette fonction lorsqu'elle n'est plus usitée, nous optimisons celles de l'utilisateur.

Un dernier paramètre (`overwrite`) permet à chaque interpolation de reprendre la main si l'utilisateur activait une entrée du menu alors qu'une autre interpolation est déjà en cours d'exécution. L'interpolation en cours est dans ce cas interrompue et permet à la nouvelle de démarrer. Cette propriété est utilisée pour éviter les conflits lorsque plusieurs interpolations ciblent les mêmes objets, ce qui est notre cas dans ce dispositif de navigation.

Propriété `overwrite` pour `TweenMax`

La propriété `overwrite` contrôle le comportement des autres interpolations si elles interviennent sur le même objet. La valeur attendue est un chiffre entier compris entre 0 et 3 :

- **0 (NONE).** Aucune interpolation n'est neutralisée. Cette option risque de créer des conflits si d'autres interpolations affectent simultanément le même objet avec les mêmes propriétés.
- **1 (ALL).** Cette valeur est celle activée par défaut sauf si la méthode `OverwriteManager.init()` est invoquée. Toutes les interpolations qui affectent le même objet sont neutralisées, qu'elles soient actives ou non. Par exemple :

```
TweenMax.to(mc, 1, {x:100, y:200});
TweenMax.to(mc, 1, {x:300, delay:2, overwrite:1});
// cette interpolation écrase la précédente.
```

- **2 (AUTO).** Cette valeur est celle activée par défaut lorsque la méthode `OverwriteManager.init()` est invoquée. La valeur 2 écrase uniquement les propriétés communes avec les interpolations en cours sur les mêmes objets. Par exemple :

```
TweenMax.to(mc, 1, {x:100, y:200});
TweenMax.to(mc, 1, {x:300, overwrite:2});
// seule la propriété x de l'interpolation précédente est écrasée.
```

- **3 (CONCURRENT).** Interrompt toutes les interpolations, uniquement ayant cours, et qui affectent les mêmes objets. Par exemple :

```
TweenMax.to(mc, 1, {x:100, y:200});
TweenMax.to(mc, 1, {x:300, delay:2, overwrite:3});
// cette interpolation n'écrase pas la précédente, car elle intervient
// avec un délai de 2 secondes après l'autre,
// bien qu'elles affectent le même objet mc.
```

Les conditions sont déclinées pour chaque entrée. Une action trace permet également de visualiser que le ciblage des éléments fonctionne bien. Cette action vous permet aussi de personnaliser facilement ce développement en Y ajoutant vos propres actions, en rapport avec chaque lien. Il suffit alors de reprendre simplement son nom pour un ciblage dynamique, comme vu précédemment.

Dans la troisième partie, un gestionnaire ENTER_FRAME recalcule en la position de chaque entrée de menu :

```
addEventListener(Event.ENTER_FRAME, remplacerEntreesMenu);
function remplacerEntreesMenu(evt:Event) {
    menu_mc.entree2_mc.y=initEntree2Y+(menu_mc.entree1_mc.sousMenu_mc.
        ➤ y-initSousMenu1Y);
    menu_mc.entree3_mc.y=initEntree3Y+(menu_mc.entree1_mc.sousMenu_mc.
        ➤ y-initSousMenu1Y)+(menu_mc.entree2_mc.sousMenu_mc.y-initSousMenu2Y);
}
```

La première instruction détermine la position Y de la deuxième entrée. La première, demeurant immobile, il n'est donc pas utile de la calculer.

Dans cette première équation, nous spécifions que la position en Y de l'entrée 2 doit dépendre de la position en Y du bas du sous-menu de l'entrée 1. Le bas du sous-menu est calculé en additionnant sa hauteur à sa position courante en Y (voir Figure 13.18).

Le principe est le même pour l'entrée 3, à la différence que nous décalons le calcul d'une ligne, en ajoutant la position courante en Y du deuxième sous-menu.

À retenir

- Pour réaliser des menus qui se repositionnent les uns par rapport aux autres, dynamiquement, nous devons utiliser un gestionnaire ENTER_FRAME qui définit la position de chaque entrée de menu par rapport à la position courante du sous-menu qui la précède.
- Le positionnement des sous-entrées de menu est important, dans la construction du document, car cela détermine leur visibilité lorsqu'elles sont déployées.

Activer et désactiver les boutons et les MovieClip

Dans bien des cas, il est nécessaire de désactiver des liens et des MovieClip initialement associés à un écouteur d'événement. Lorsque vous chargez une rubrique par-dessus une interface déjà riche en fonctionnalités, par exemple, il peut être conflictuel de garder actifs certains éléments laissés en arrière-plan.

Dans cette section, nous présentons les différentes méthodes employées pour neutraliser un bouton ou un MovieClip. Mais aussi pour matérialiser le pointeur en forme de main sur les MovieClip et ainsi améliorer l'ergonomie de certaines mises en scène.

- Pour désactiver uniquement les états des boutons (haut, dessus et abaissé), utilisez l'instruction : `nomDuBouton.enabled=false`;
- Pour désactiver l'interactivité d'un bouton ou d'un MovieClip, utilisez : `nomDuBoutonOuClip.mouseEnabled=false`;

- Pour désactiver l'interactivité d'un bouton ou d'un MovieClip ainsi que tout élément enfant de chaque objet géré avec `evt.target` (précision valable pour les clips ou les Sprites) : `nomDuBoutonOuClip.mouseChildren=false;`
- Pour activer uniquement l'affichage de la main, au survol du pointeur, sur un MovieClip, choisissez l'instruction : `nomDuClip.buttonMode = true;`
- Pour désactiver l'affichage de la main sur un symbole bouton, prenez l'instruction suivante : `nomDuBouton.useHandCursor=false;`

Vous pouvez aussi neutraliser les interactions avec l'une de ces méthodes :

- Masquer un bouton ou un MovieClip à l'aide de la propriété `visible`, passée sur `false`.
- Supprimer l'objet de la liste d'affichage avec la méthode `removeChild()`.
- Supprimer l'écouteur associé à l'objet bouton ou MovieClip avec la méthode `removeEventListener`.

Attention, ces techniques ne suppriment pas les actions éventuellement en cours d'exécution, elles rendent seulement les objets interactifs indisponibles pour le lancement de nouvelles actions.



Ciblage des boutons entre SWF imbriqués. Il est souvent utile de pouvoir neutraliser une action ou réagir en fonction d'une action lancée dans un document SWF importé. Reportez-vous au Chapitre 8 pour en savoir plus sur les techniques de ciblage d'objets dans des contenus imbriqués.

Target et currentTarget. La définition des actions sur les symboles boutons et MovieClip se détermine aussi dans le cadre d'éléments contenus à l'intérieur d'un objet parent et ciblés avec les propriétés `target` et `currentTarget`. Reportez-vous également au Chapitre 5 pour en savoir plus à ce sujet.

Synthèse

Dans ce chapitre, vous avez appris à développer des systèmes de navigation sophistiqués en y intégrant des boutons avec des états visités ou activés. Vous avez appris à transformer de simples liens en zone interfaçable et interactive sans conflit avec l'environnement initial. Vous avez également vu comment gérer la superposition dynamique de menus déroulants. Vous avez enfin appris à réaliser un dispositif de navigation global concentré dans une fenêtre réduite. Vous êtes en mesure de réaliser des interfaces souples et ergonomiques.

14

La communication Flash/HTML

Introduction

Interprété la plupart du temps depuis une fenêtre de navigateur, Flash offre de nombreuses fonctionnalités qui favorisent l'échange entre des contenus HTML et Flash. Vous pouvez gérer l'historique de navigation de Flash depuis le navigateur. Vous pouvez aussi appeler une configuration particulière d'un document Flash, en fonction de la page HTML qui y fait référence, en invoquant par exemple directement une image-clé du scénario. Naturellement, vous pouvez également lancer des pages HTML à partir de Flash. Enfin, pour offrir encore plus de fonctionnalités dans l'interface de Flash, et rappeler certaines options de la fenêtre de navigateur, le menu contextuel de Flash est personnalisable et vous pouvez y attacher les instructions de votre choix. Enfin, vous pouvez, en ajoutant un paramètre dans la page HTML, configurer une animation Flash pour rendre la scène transparente.

Dans ce chapitre, nous allons aborder l'ensemble de ces mécanismes pour vous permettre d'intégrer efficacement vos réalisations au sein d'un environnement web HTML. À l'issue de ce chapitre, vous serez capable d'élaborer facilement des liaisons entre différents types de contenus, réalisés en HTML et en Flash.

Menu contextuel du lecteur Flash

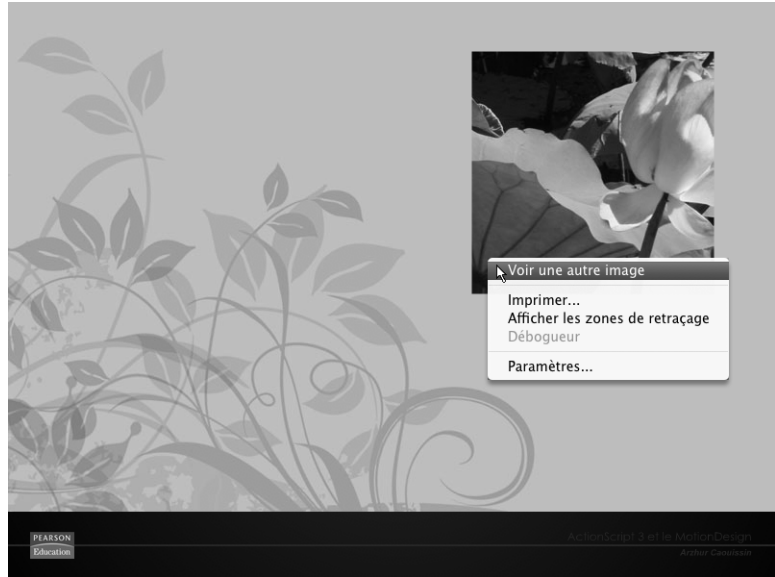
Un menu contextuel est accessible par le clic-droit de la souris sur la fenêtre de l'animation. Si votre souris ne dispose pas du clic-droit, le menu contextuel peut apparaître en effectuant simultanément un clic de souris et en appuyant sur la touche Contrôle du clavier.

La classe `contextMenu` de Flash permet de personnaliser une partie du menu contextuel. L'option d'affichage de la version du lecteur reste cependant affichée. Vous pouvez aussi choisir de conserver certaines des options disponibles par défaut, comme l'option Imprimer, par exemple.

Dans cette section, nous allons voir comment personnaliser un menu contextuel sur un symbole de type `MovieClip`. Nous allons permettre à l'utilisateur d'afficher une nouvelle image, en sélectionnant une option du menu contextuel à l'emplacement d'une photo, pour ce faire (voir Figure 14.1).

Figure 14.1

Aperçu de la scène principale.



Exemples > ch14_communicationFlashHTML_1 fla

Dans le document "ch14_communicationFlashHTML_1 fla", dans la scène principale, un symbole `vegetal_mc` décore la scène. Au-dessus, un MovieClip nommé `galerie_mc` affiche une suite d'images.

Dans le document SWF si l'utilisateur effectue un clic-droit sur l'image affichée, une option propose de voir une autre image. Si l'option est activée, une nouvelle image apparaît aussitôt.

Dans la fenêtre de scénario de la scène principale, le calque Actions affiche le code suivant :

```
//----- actions
galerie_mc.stop();

//----- Personnalisation du menu contextuel
// attacher le menu à un objet
var menuContextuel:ContextMenu= new ContextMenu();
galerie_mc.contextMenu=menuContextuel;

// purger le contenu par défaut
menuContextuel.hideBuiltInItems();
var entreesParDefaut:ContextMenuBuiltInItems=menuContextuel.builtInItems;
entreesParDefaut.print=true;

// ajouter les entrées du menu
var entree1:ContextMenuItems=new ContextMenuItem("Voir une autre image");
menuContextuel.customItems.push(entree1);
entree1.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, fonctionEntree1);

// attacher les fonctions aux entrées
```

```
function fonctionEntree1(evt:ContextMenuEvent) {
    if (galerie_mc.currentFrame==galerie_mc.totalFrames) {
        galerie_mc.gotoAndStop(1);
    } else {
        galerie_mc.nextFrame();
    }
}
```

Pour créer un menu contextuel, nous devons d'abord purger les entrées existantes. C'est alors que nous ajoutons de nouveaux éléments auxquels nous associerons des fonctions.

Dans notre exemple, la première instruction interrompt la tête de lecture sur la première image du symbole galerie_mc :

```
//----- actions
galerie_mc.stop();
```

Nous attachons ensuite un nouvel objet menuContextuel à notre galerie. Nous pouvons, ainsi, spécifier un menu différent pour chaque conteneur, si nous le souhaitons :

```
// attacher le menu à un objet
var menuContextuel:ContextMenu= new ContextMenu();
galerie_mc.contextMenu=menuContextuel;
```

Puis, nous purgeons le menu actuel de ses entrées affichées par défaut.

```
// purger le contenu par défaut
menuContextuel.hideBuiltInItems();
```

Nous utilisons ici plusieurs méthodes. La première, `hideBuiltInItem()` supprime tous les éléments sauf les paramètres du lecteur Flash. Il est possible de conserver néanmoins certaines options. Pour cela, il suffit de les ajouter à la liste vidée. Pour ajouter une option par défaut, nous typons un nouvel objet `ContextMenuBuiltInItem` auquel nous spécifions l'option à intégrer :

```
var entreesParDefaut:ContextMenuBuiltInItems=menuContextuel.builtInItems;
entreesParDefaut.print=true;
```

Les options disponibles par défaut sont : `forwardAndBack` (affiche une image en arrière ou en avant), `loop` (boucle), `play` (lire), `print` (imprimer), `quality` (qualité), `rewind` (rembobiner), `save` (enregistrer) et `zoom` (zoom). Pour les activer, il suffit de les attacher au menu désigné et de les passer en `true`, comme dans notre exemple avec l'option d'impression `print`.

Une fois la structure de base définie, nous pouvons personnaliser le menu en y intégrant nos propres entrées :

```
// ajouter les entrées du menu
var entree1:ContextMenuItems=new ContextMenuItem("Voir une autre image");
menuContextuel.customItems.push(entree1);
```

Pour ajouter une entrée, nous créons un nouvel objet `ContextMenuItems()` et renseignons, en paramètre, une chaîne de caractères qui correspond au texte à afficher dans le menu.

Pour ajouter cet objet au menu, nous l'implémentons comme pour l'ajouter à un tableau, à l'aide de la méthode `push()`. Attention, l'objet doit être implémenté en tant qu'entrée – élément d'une liste. Nous devons pour cela préciser la propriété `customItems`, juste avant d'appliquer la méthode `push()`.

Vous pouvez ajouter autant d'entrées que voulu. Créez alors autant de nouveaux objets `ContenuMenuItem`.

Lorsque les entrées ont été ajoutées, il ne reste plus qu'à leur affecter une action. Nous utilisons, pour établir cette relation, un écouteur avec l'événement `MENU_ITEM_SELECT` qui désigne le moment où l'utilisateur sélectionne l'entrée du menu :

```
entree1.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, fonctionEntree1);
```

Puis, nous développons les actions appelées. Dans notre exemple, la fonction affiche une nouvelle image de la galerie, avec la méthode `nextFrame()`. Si cette image est la dernière de la séquence animée, alors, la tête de lecture revient à la première image. Elle boucle :

```
// attacher les fonctions aux entrées
function fonctionEntree1(evt:ContextMenuEvent) {
    if (galerie_mc.currentFrame==galerie_mc.totalFrames) {
        galerie_mc.gotoAndStop(1);
    } else {
        galerie_mc.nextFrame();
    }
}
```

À retenir

- Pour créer un menu contextuel personnalisé, nous utilisons la classe `ContextMenu`.
- Les options du menu contextuel de Flash, affichées par défaut, peuvent être supprimées ou partiellement conservées. Seule la version du lecteur ne peut être supprimée du menu contextuel.

Navigation Flash vers le HTML

La base de la communication entre un contenu en Flash et une page HTML est la création de liens. Ces liens appellent une page HTML, un site distant ou activent l'ouverture d'un logiciel de messagerie, par exemple.

Dans cette section, nous présentons la syntaxe de création de lien dans Flash pour appeler une page HTML et d'autres types de références hypertextes (sites, messagerie, PDF).

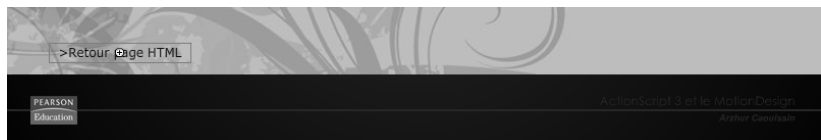


Exemples > ch14_communicationFlashHTML_2 fla

Sur la scène principale du document "ch14_communicationFlashHTML_2 fla" figure un bouton affichant un texte simple : ">Retour à la page HTML" (voir Figure 14.3).

Figure 14.3

Aperçu de la scène principale.



Dans le fichier SWF, en cliquant sur le bouton, une page HTML est bien appelée ; page dont nous détaillons d'ailleurs le contenu dans la section suivante.

Dans la fenêtre de scénario de la scène principale, sur le calque Actions, nous pouvons lire le code suivant :

```
// lien HTML
lienHTML_btn.addEventListener(MouseEvent.CLICK,lienHTML);
function lienHTML (evt:MouseEvent) {
    navigateToURL(new URLRequest("ch14_communicationFlashHTML_3b.html"));
}
```

En ActionScript 3, pour appeler une page HTML, nous utilisons la méthode `navigateToURL` et spécifions, en paramètre, un objet `URLRequest` qui fait référence à une adresse définie sous la forme d'une chaîne de caractères.

L'adresse utilisée peut appeler une page web locale, au sein du même site que la page qui contient le document Flash. Les exemples suivants invoquent différents éléments :

- une page de contact au format PHP :

```
navigateToURL(new URLRequest("contact.php"));
```

- une adresse de site web distant :

```
navigateToURL(new URLRequest("http://www.pearson.fr/"));
```

- l'ouverture d'une fenêtre de messagerie :

```
navigateToURL(new URLRequest("mailto:contact@pearson.fr?subject=demande de  
➔ renseignement"));
```

- l'ouverture d'un PDF :

```
navigateToURL(new URLRequest("nomDuDocument.pdf"));
```

- Pour appeler un site dans une nouvelle fenêtre, nous ajoutons la cible :

```
navigateToURL(new URLRequest("http://www.pearson.fr", "_blank"));
```



Les cibles d'affichage. Pour chaque lien hypertexte créé, une cible permet de définir la manière dont le navigateur affiche la page appelée. Lorsque les sites web usaient encore, dans le passé, de structures composées de jeux de cadres (framesets), nous pouvions utiliser les valeurs `_self`, `_blank`, `_parent` ou `_top` afin de déterminer si la page appelée devait se substituer ou non aux jeux de cadres. Les liens demeurant désormais indépendants de ce type de structure, nous n'utilisons plus que `_self` ou `_blank`.

- `_self` : la cible `_self` équivaut à ne rien indiquer et signifie que la page appelée doit s'afficher dans le même cadre que celui où l'on a cliqué.
- `_blank` : la cible `_blank` appelle l'affichage dans une nouvelle fenêtre de navigateur. Le terme anglais *Blank* désigne un espace vierge, représenté ici par l'ouverture d'une nouvelle fenêtre. Ne confondez pas ce mécanisme avec le comportement JavaScript permettant d'enclencher l'ouverture d'une fenêtre pop-up, que les usages ont également proscrits. L'affichage d'une page avec `_blank` établie en outre une rupture sémantique avec la page initiale. Cette technique

d'affichage ne doit donc pas être abusée, surtout dans le contexte de création de sites qui doivent répondre aux règles élémentaires d'accessibilité (voir aussi les sections en fin d'ouvrage sur l'accessibilité).

La plupart des contenus appelés s'affichent dans la même fenêtre que le document Flash, à la place du Flash lui-même, à l'exception des scripts JavaScript ou PHP (ou autres) éventuellement invoqués. Les pages pour lesquelles une cible de type `_blank` est également associée s'affichent dans une nouvelle fenêtre de navigateur et préservent l'affichage de votre document dans la fenêtre initiale.

Attention, cette option n'est recommandée que pour les liens faisant appel à des éléments situés à une autre URL, car elle entraîne une rupture sémantique du flux qui peut perturber certains utilisateurs.

Lorsque vous désignez une adresse d'un site ou d'une page distante, veillez à toujours commencer l'adresse par `http://`.

Si les liens créés appellent des instructions localisées dans la page HTML qui contient le Flash, ou lorsque des actions ne peuvent être lancées que par un navigateur, pour tester le bon fonctionnement des liens hypertextes, il est recommandé de projeter la page dans un navigateur et non de vérifier les animations seulement depuis Flash en publiant le fichier SWF. Le lecteur Flash, activé à la publication du document, n'est pas un navigateur. Ainsi, il ne peut exécuter tous les types de liens (notamment pour l'ouverture de la fenêtre de messagerie ou lancer les scripts exécutés localement ou depuis un serveur distant).



Créer un lien HTML sans ActionScript dans Flash. Si vous cherchez à réaliser un lien dans le flux d'un texte, sans style spécifique, sans avoir à créer de symbole Bouton ou MovieClip et sans avoir à coder en ActionScript, vous pouvez utiliser les propriétés de texte. Sélectionnez d'abord la portion de texte à rendre cliquable. Et depuis l'inspecteur de propriétés, dans le champ Lien (catégorie Option), saisissez directement le lien ou le type de référence hypertexte à appeler. Puis, appuyez sur Entrée pour confirmer la saisie.

Il est également possible de générer un lien HTML dynamiquement si le champ de texte dynamique affiche un rendu HTML. En ActionScript, modifiez la valeur du champ de texte en spécifiant, dans la chaîne de caractères du texte à afficher, la balise HTML qui permet de créer un lien. Le texte dynamique de nom d'occurrence `monTexte_txt` utilise, dans l'exemple suivant, cette propriété :

```
monTexte_txt.htmlText = '<a href="http://www.pearson.fr">Lien</a>'
```

À retenir

- Vous pouvez appeler tout type de contenu extérieur à Flash et l'afficher dans la fenêtre de navigateur à la place du Flash en utilisant la méthode `navigateToURL` à laquelle vous associez un nouvel objet `URLRequest` qui affiche la référence hypertexte mentionnée.
- Il est préférable de tester les liens dans le contexte du navigateur, voire en ligne, lorsque ces liens se réfèrent à des comportements dynamiques exécutés côté serveur ou par le navigateur.

Navigation HTML vers Flash

Si vous réalisez un site presque entièrement en Flash et que, ponctuellement, vous appelez une page HTML, vous aimeriez certainement pouvoir faire un lien de retour depuis cette page, vers un endroit précis de votre document Flash, et pas forcément vers la première image du scénario de votre document Flash. Cela est possible en adoptant une syntaxe particulière pour les liens placés dans la page HTML et une structure adaptée pour la construction du document Flash.

Dans ce contexte, le lien HTML fait référence, en fait, à une image clé du scénario du document Flash en ajoutant, au nom de la page qui contient le Flash, un dièse (#), suivi du nom d'étiquette de l'image cible :

nomDeLaPage.html#nomDeLetiquette

Dans cette section, nous présentons un document SWF structuré en vue d'être appelé par un lien HTML. Le lien HTML contient, de son côté, des liens qui activent distinctement chacune des rubriques du Flash.

Cette technique impose une conception de site dans le scénario, pour lequel chaque type de contenu est réparti, à un endroit distinct des autres.



Exemples > ch14_communicationFlashHTML_3 fla

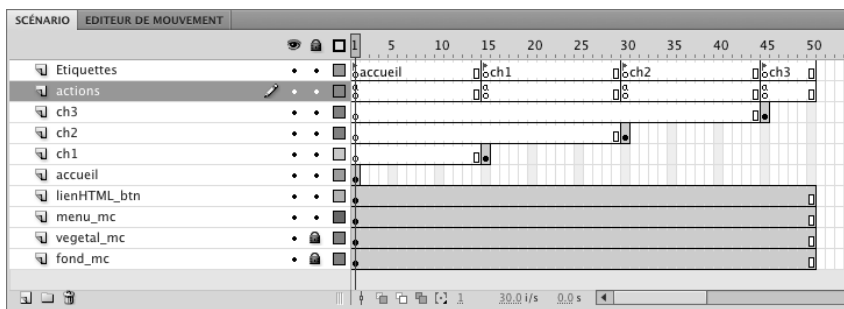
Exemples > ch14_communicationFlashHTML_3.html

Exemples > ch14_communicationFlashHTML_3b.html

Dans notre exemple, nous disposons d'un document Flash dont l'interface est distribuée sur le scénario en mode séquentiel (voir Figure 14.4). Chaque emplacement d'étiquette correspond à une nouvelle rubrique.

Figure 14.4

Aperçu
du scénario
de la scène
principale.



Ce document est placé dans la page HTML "ch4_communicationFlashHTML_3.html". En affichant ce fichier dans le navigateur, nous constatons dans un premier temps que la page affiche bien l'animation (voir Figure 14.5).



Pour cet exemple, ne publiez pas le document SWF en faisant F12 ou Fichier > Publier. Vous risqueriez d'écraser la page HTML déjà mise en forme et qui porte le même nom que le document Flash. Pour publier le document SWF, faites simplement Cmd+Entrée (Mac) ou Ctrl+Entrée (Windows). Pour projeter la page HTML directement dans le navigateur, lancez-la depuis le système d'exploitation.

Figure 14.5

Aperçu dans le navigateur.



En cliquant sur les liens du menu situés à gauche, vous accédez aux différentes rubriques mises en place dans le scénario de Flash (voir Figures 14.6 à 14.8).

Figure 14.6



En cliquant sur le lien HTML situé en bas du document Flash, vous accédez à une page HTML (vue à la section précédente). Cette page HTML affiche une série de liens codés en HTML.

Figure 14.7

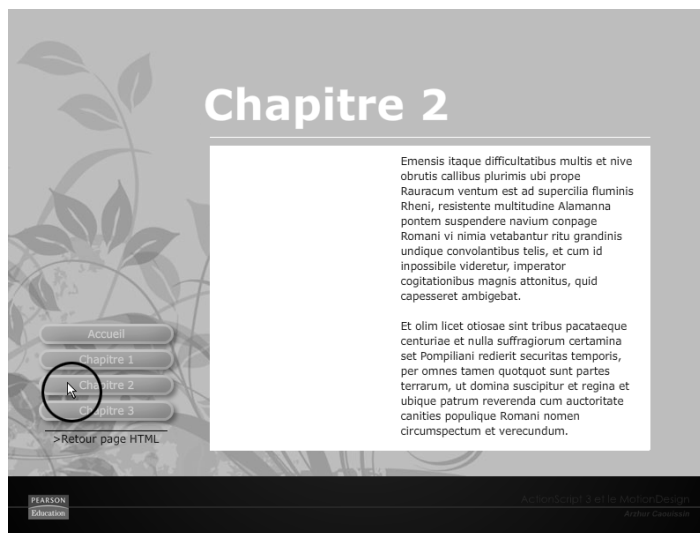
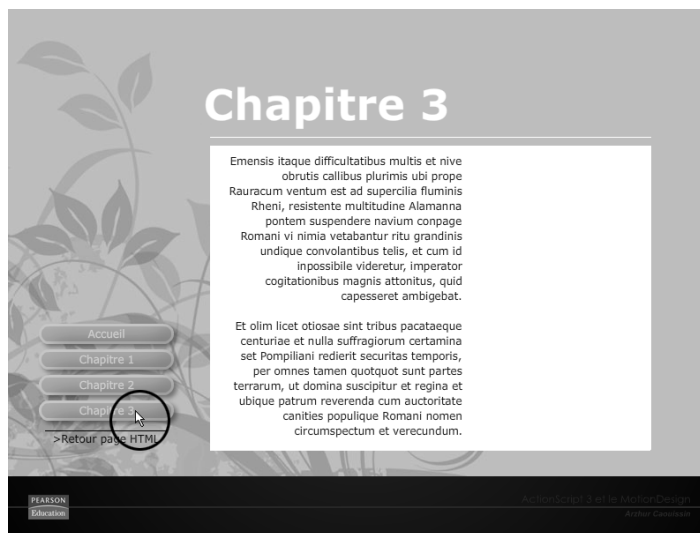


Figure 14.8



En cliquant sur chaque lien, vous atteignez directement l'image du scénario qui correspond à la rubrique activée (voir Figure 14.9).

Dans le document Flash, sur la scène principale, se trouve un symbole nommé menu_mc. Dans ce symbole, nous pouvons distinguer quatre boutons respectivement nommés accueil, ch1, ch2 et ch3 (voir Figure 14.10).

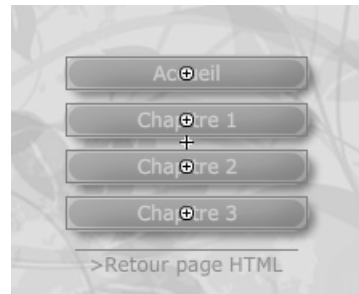
Dans le scénario de la scène principale, chaque rubrique est placée précisément sous une étiquette. L'accueil est sous l'étiquette accueil. Le Chapitre 1 est sous l'étiquette ch1, et ainsi de suite pour les Chapitres 2 et 3 (voir Figure 14.4). Vous relevez que les noms d'étiquettes sont identiques aux noms d'occurrence empruntés pour les boutons du menu.

Figure 14.9

Aperçu de la page HTML appelée.

**Figure 14.10**

Boutons de navigation dans le Flash.



Cela n'a pas d'incidence directe sur la navigation depuis la page HTML, mais simplifie le codage au sein du document Flash pour une navigation interne.



Placer un nom d'étiquette sur une image du scénario. Pour placer un nom d'étiquette sur une image du scénario, vous devez d'abord ajouter une image clé (de préférence vide), sur un calque dédié à cela dans le scénario, à l'endroit ciblé. Puis, cliquez sur cette image du scénario pour en afficher les propriétés. Dans l'inspecteur de propriétés, attribuez un identifiant sans caractères spéciaux, sans espace, et ne commençant pas par un chiffre. Appuyez sur la touche Entrée pour confirmer la saisie.

Dans le scénario de la scène principale, sur le calque Actions, à l'image 1, le code affiché est le suivant :

```
stop();

menu_mc.addEventListener(MouseEvent.CLICK,afficherRubrique);
function afficherRubrique (evt:MouseEvent) {
    gotoAndStop(evt.target.name);
}

lienHTML_btn.addEventListener(MouseEvent.CLICK,lienHTML);
function lienHTML (evt:MouseEvent) {
    navigateToURL(new URLRequest("ch14_communicationFlashHTML_2b.html"));
}
```

Tout d'abord, une fonction permet de naviguer à l'intérieur du document Flash en cliquant sur les liens contenus dans le symbole menu_mc. Cette fonction utilise la propriété target qui établit une relation directe entre le nom de l'occurrence activée et le nom de l'étiquette à atteindre. Si nous cliquons sur le bouton ch1, son nom est donc directement passé en paramètre de la méthode gotoAndStop() et nous conduit à cette image du scénario. Cette technique nous épargne l'ajout de structures conditionnelles, pour simplement organiser une navigation dans le scénario.

En dessous, figure le lien qui pointe vers la page HTML – nous l'avons utilisé dans la section précédente.

Ce document Flash est donc autonome. Aucune relation ne l'attache à la page HTML qui s'y réfère. Mais, ce qui permet aux liens de la page HTML de cibler la bonne rubrique, c'est qu'il est structuré avec des étiquettes dont les noms sont repris dans les liens de la page HTML.

Si nous revenons dans la page HTML qui porte le nom "ch14_communicationFlashHTML_3b.html", dans Dreamweaver par exemple ou directement dans le code source du document, en sélectionnant chaque lien individuellement, vous remarquez la valeur inscrite dans l'attribut href .

Pour appeler la rubrique accueil, le lien HTML fait référence d'abord à la page qui contient le Flash (ch14_communicationFlashHTML_3.html), suivi directement d'un dièse (#) et de l'étiquette (accueil). Ce qui donne :

```
<a href="ch14_communicationFlashHTML_3.html#accueil">texte du lien</a>
```

Le principe est décliné pour chaque lien de la page HTML.

Mais, dans ce mécanisme relativement simple, nous devons souligner quelques contraintes.

Lorsque le lien de la page HTML atteint directement une image du scénario, les écouteurs, placés sur la première image du scénario, ne sont plus actifs. Vous devez recopier chaque écouteur placé sur la première image au niveau de chaque étiquette. Les fonctions, elles, restent actives. Il n'est pas nécessaire de les dupliquer. D'autant qu'il ne peut y avoir deux fonctions de même nom. Dans le calque Actions, à l'image ch1, ch2 et ch3, nous pouvons lire le code suivant :

```
stop();  
menu_mc.addEventListener(MouseEvent.CLICK,afficherRubrique);
```

Nous avons un stop qui empêche la tête de lecture de poursuivre l'animation. Nous pouvons également lire une nouvelle instance de l'écouteur déjà placé à l'image 1.



Tester les liens HTML sur un serveur distant. Avec certaines configurations, il est possible que les liens HTML qui appellent une étiquette Flash ne soient pas directement actifs sur votre poste de travail. Placez dans ce cas vos fichiers sur un serveur distant, et exécutez à nouveau les pages. Les liens sont bien actifs.

À retenir

- Il est possible de cibler directement une image du scénario dans Flash, à partir d'un lien codé en HTML. Pour cela, nous invoquons un nom d'étiquette (#nomEtiquette) directement depuis le lien HTML.

Historique de l'animation Flash dans le navigateur

Le bouton Suivant et Retour du navigateur peuvent contrôler l'affichage des contenus dans un document Flash. Pour cela, le site www.asual.com met à disposition les classes SWFAddress et SWFObject, gratuitement et en téléchargement libre, à l'adresse suivante : <http://www.asual.com/swfaddress/>.



Installer une classe importée. Pour voir comment installer la classe nativement ou ponctuellement, reportez-vous au début du Chapitre La 3D et PaperVision. Nous nous contentons, pour cet exemple, de citer le site référent. Une documentation détaillée et un forum sont disponibles sur leur site Internet.

Ces classes fonctionnent sur le même principe que le mécanisme des liens d'une page HTML vers un contenu Flash, évoqué dans la section précédente. À la différence que les liens ne sont pas activés par la page HTML, mais par un script JavaScript interprété par le navigateur.

Pour les utiliser, vous devez donc créer un document Flash qui sera contenu dans une page HTML, laquelle fait référence à plusieurs fichiers JavaScript. Le code proposé par le site www.asual.com pour intégrer le contenu Flash est le suivant :

```
<script type="text/JavaScript" src="swfobject/swfobject.js"></script>
<script type="text/JavaScript" src="swfaddress/swfaddress.js"></script>
<script type="text/JavaScript">swfobject.embedSWF('website.swf', 'website',
'100%', '100%', '9','expressInstall.swf', {}, {menu: 'false'}, {id: 'website'});
</script>
```

Dans Flash, vous devez ensuite placer les commandes dans le scénario. Elles seront automatiquement interprétées par le même JavaScript utilisé dans la page HTML. Dans le document Flash, sur la première image du scénario de la scène principale, le site www.asual.com propose les actions suivantes :

```
// SWFAddress actions
function btnClick(e:MouseEvent) {
    SWFAddress.setValue(e.target.deepLink);
}
function btnRollOver(e:MouseEvent) {
    SWFAddress.setStatus(e.target.deepLink);
}
function btnRollOut(e:MouseEvent) {
    SWFAddress.resetStatus();
}

// SWFAddress handling
function handleSWFAddress(e:SWFAddressEvent) {
    try {
        if (currentFrame == 2 && e.value == '/') {
            play();
        } else {
            gotoAndStop('$' + e.value);
        }
    } catch(err) {
        gotoAndStop('$/error/');
    }
}
```

```

var title:String = 'SWFAddress Website';
for (var i = 0; i < e.pathNames.length; i++) {
    title += ' / ' + e.pathNames[i].substr(0,1).toUpperCase() + e.path-
    ➤ Names[i].substr(1);
}
SWFAddress.setTitle(title);
}
SWFAddress.addEventListener(SWFAddressEvent.CHANGE, handleSWFAddress);
stop();

```

Nous distinguons les différentes fonctions faisant référence à des occurrences de MovieClip distribuées dans le scénario à chaque niveau d'étiquette. Ces fonctions sont donc valides pour toute la durée du scénario. La variable `deepLink` est simplement mise à jour en fonction de l'objet activé (target). C'est donc l'étiquette invoquée qui est différente en fonction de l'objet activé.

Dans le scénario, à chaque nouvelle rubrique, utilisez un nom d'étiquette qui reprend cette syntaxe:

```
$/portfolio/
```

Ce nom commence avec le caractère dollar. Puis, un nom, sans caractères spéciaux ni accent est placé entre deux slashes.

À l'intérieur de chaque MovieClip – qui est utilisé comme bouton et dont la navigation doit être contrôlée par JavaScript –, www.asual.com propose d'utiliser le code suivant :

```

this.deepLink = '/about/';
this.buttonMode = true;
this.addEventListener(MouseEvent.CLICK, (parent as MovieClip).btnClick);
this.addEventListener(MouseEvent.ROLL_OVER, (parent as MovieClip).btnRollOver);
this.addEventListener(MouseEvent.ROLL_OUT, (parent as MovieClip).btnRollOut);

```

Dans ce code, nous relevons principalement l'affectation d'une nouvelle valeur pour la variable `deepLink`, qui véhicule la cible à atteindre par le bouton cliqué. Cette valeur, enregistrée par JavaScript, va permettre au navigateur de contrôler toute la navigation à l'intérieur du document Flash.

Enfin, dans le dossier de votre document Flash, vous devez placer les éléments suivants :

- la classe intitulée `SWFAddress.as` ;
- la classe intitulée `SWFAddressEvent.as` ;
- le dossier de scripts `swfaddress/` ;
- et le dossier de scripts `swfobject/`.

Publiez le document SWF. En activant les liens du menu, le document Flash défile normalement. Mais les commandes de l'historique du navigateur enregistrent votre navigation. En activant les boutons Suivant et Retour, vous pouvez revenir dans le document Flash comme si vous naviguiez de page HTML en page HTML.

Pour utiliser et tester un fichier Flash dans un environnement déjà mis en forme, vous pouvez télécharger le dossier compressé de présentation déjà structuré, et disponible directement en ligne. Pour des raisons liées à la protection des droits d'auteurs, nous ne

pouvons pas inclure ces ressources dans notre ouvrage. Vous pouvez en revanche les télécharger librement à l'adresse suivante : <http://sourceforge.net/projects/swfaddress/files/swfaddress/SWFAddress%202.4/swfaddress-2.4.zip/download>. Dans le répertoire téléchargé, différentes configurations sont proposées. Pour un développement en ActionScript 3, utilisez les éléments du répertoire CS3 mis à disposition dans le dossier Samples.

À retenir

- Il est possible d'activer les boutons Retour et Suivant du navigateur pour créer un historique de la navigation dans Flash. Pour cela, nous utilisons la classe SWFAddress et SWFObject disponible gratuitement sur le site www.asual.com.

Importer du HTML dans un SWF

Plus qu'une simple fonctionnalité, l'intégration de contenus formatés avec des balises HTML est une solution presque incontournable quand il s'agit d'intégrer des contenus éditoriaux. Sans avoir à recourir à des systèmes de gestion de contenu, il demeure assez facile d'importer et de mettre en forme des textes et des images avec des feuilles de style. Pour cela, nous utilisons un champ de texte dynamique et, éventuellement, un composant ScrollBar (ou un ascenseur fait maison, comme vu au Chapitre 2).

Dans cette section, nous utilisons une interface en Flash qui charge un fichier texte et une feuille de style au format CSS (voir Figure 14.11).

Figure 14.11

Aperçu du document publié.



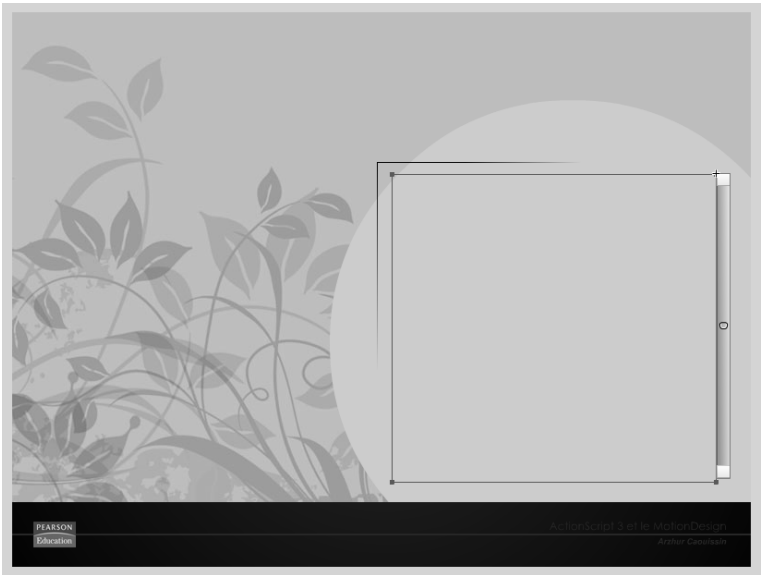


Exemples > ch14_communicationFlashHTML_4 fla
Exemples > html > styles.css
Exemples > html > texte.txt

Dans le document "ch14_communicationFlashHTML_4 fla", à droite de la scène principale, est placé un aplat de couleur (forme graphique). Au-dessus, un champ de texte dynamique porte le nom d'occurrence contenu_txt. À sa droite, un composant ScrollBar porte le nom d'occurrence ScrollBar (voir Figure 14.12).

Figure 14.12

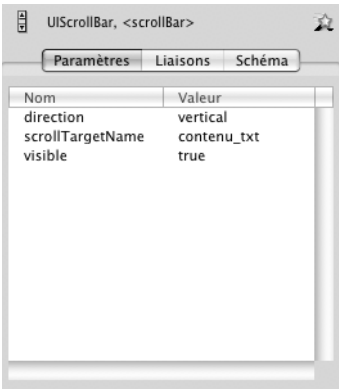
Aperçu du document.



Dans l'inspecteur de composants (Fenêtre > Inspecteur de composants), la propriété scrollTargetName, qui définit le champ de texte à contrôler, fait référence au nom de notre champ de texte dynamique (voir Figure 14.13).

Figure 14.13

Fenêtre
Inspecteur de
composants.



En dehors du document Flash, dans le dossier html/, placé dans le répertoire des exemples du livre, nous disposons d'un fichier nommé `texte.txt` qui affiche en partie le code suivant :

```
content=<p class="titre">A la une</p><p><i>de Arzhur CAOUISSIN</i><br><li><b>
Source :</b> Les marais salant de Guérande</li><li><b>Site Web :</b> <a href=
"http://www.ot-guerande.fr/" target="_blank"><span class="lien">http://www.ot-
guerande.fr</span></a></li></p><p>Lorem ipsum dolor sit amet, consectetuer
adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore
magna aliquam erat volutpat. Ut wisi enim ad minim veniam, .../</p>
```

Une variable du nom de content ouvre sur une série de balises HTML qui organisent le flux d'un texte et contiennent, parfois, des références à des classes CSS (`.titre`, `.lien`) ou à des balises d'apparence modifiée, dans la feuille de styles (`p`). Les images importées sont, elles, déjà placées localement dans le répertoire images du dossier des exemples du livre.

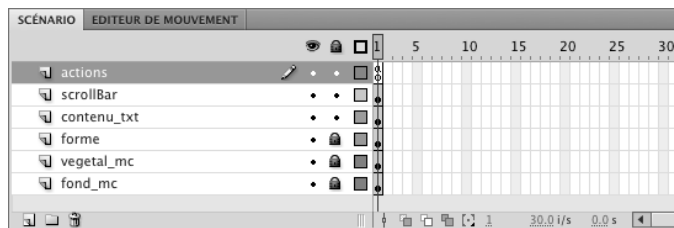
Le dossier html/ des exemples du livre accueille cette feuille de style qui adopte la forme suivante :

```
p {
font-family: Arial,Helvetica,sans-serif;
font-size: 12px;
color: #001622;
}
.titre {
font-family: Arial,Helvetica,sans-serif;
font-size: 24px;
color: #001622;
}
.lien {
font-family: Arial,Helvetica,sans-serif;
font-size: 12px;
color: #001622;
text-decoration: underline;
}
```

Dans le scénario du document Flash, le composant Scrollbar et le texte dynamique sont répartis entre les formes d'habillage graphique et le calque Actions (voir Figure 14.14).

Figure 14.14

Aperçu du scénario de la scène principale.



Le calque Actions affiche le code suivant :

```
// Charge les CSS
var chargeurCSS:URLLoader = new URLLoader();
chargeurCSS.load(new URLRequest("html/styles.css"));
chargeurCSS.addEventListener(Event.COMPLETE, CSSchargees);
```

```
// Applique les CSS
function CSSchargees(evt:Event) {
    var stylesCSS:StyleSheet = new StyleSheet();
    stylesCSS.parseCSS(evt.currentTarget.data);
    contenu_txt.styleSheet=stylesCSS;
    chargerLeTexte();
}

// Charge le texte
function chargerLeTexte() {
    var chargeurTexte:URLLoader = new URLLoader();
    chargeurTexte.load(new URLRequest("html/texte.txt"));
    chargeurTexte.addEventListener(Event.COMPLETE, placerLeTexte);
}

// Place le texte
function placerLeTexte(evt:Event) {
    var cheminVariable:URLVariables=new URLVariables(evt.currentTarget.data);
    contenu_txt.condenseWhite=true;
    contenu_txt.htmlText=String(cheminVariable.content);
}
```

Dans cette application, nous activons le chargement des contenus textes et CSS en quatre temps. D'abord, nous chargeons les styles. Puis, nous les appliquons au conteneur du futur texte. Une fois les styles appliqués, nous pouvons importer le texte dans ce champ dynamique.

La première étape charge la feuille de style :

```
// Charge les CSS
var chargeurCSS:URLLoader = new URLLoader();
chargeurCSS.load(new URLRequest("html/styles.css"));
chargeurCSS.addEventListener(Event.COMPLETE, CSSchargees);
```

Il s'agit d'un chargeur tout simple qui exécute une fonction une fois le chargement complété. La fonction appelée applique ensuite les styles, sur le champ de texte dynamique :

```
// Applique les CSS
function CSSchargees(evt:Event) {
    var stylesCSS:StyleSheet = new StyleSheet();
    stylesCSS.parseCSS(evt.currentTarget.data);
    contenu_txt.styleSheet=stylesCSS;
    chargerLeTexte();
}
```

Dans cette fonction, nous définissons, dans un premier temps, un objet feuille de style qui va stocker les styles avant de les appliquer. Nous compilons ensuite les balises CSS avec un parseur CSS interne à Flash, pour que le code soit correctement interprété par Flash. Les données chargées (`evt.currentTarget.data`) sont passées en paramètre du parseur CSS. Le parseur affecte les styles à l'objet `StyleSheet` que nous venons d'instancier.

Puis, nous appliquons la feuille de style sur le champ de texte dynamique, à l'aide de la propriété `styleSheet`.

La fonction qui charge le texte est ensuite appelée :

```
// Charge le texte
function chargerLeTexte() {
    var chargeurTexte:URLLoader = new URLLoader();
    chargeurTexte.load(new URLRequest("html/texte.txt"));
    chargeurTexte.addEventListener(Event.COMPLETE, placerLeTexte);
}
```

Une fois le texte complété, la fonction appelée place le texte formaté dans le champ de texte dynamique :

```
// Place le texte
function placerLeTexte(evt:Event) {
    var cheminVariable:URLVariables=new URLVariables(evt.currentTarget.data);
    contenu_txt.condenseWhite=true;
    contenu_txt.htmlText=String(cheminVariable.content);
}
```

Dans cette dernière fonction, nous créons un objet de traitement de variable `URLVariables`, afin de contenir l'ensemble du code qui se situe, à l'intérieur du fichier `texte.txt`, après le signe égal. Cette variable appelle en paramètre le texte que nous venons de charger pour s'en approprier naturellement le contenu.

La propriété `condenseWhite` permet de supprimer les espaces blancs redondants, comme le fait tout interpréteur HTML.

La dernière instruction ajoute le texte chargé, dans le champ dynamique situé sur la scène, à l'aide de la propriété `htmlText`. Mais il le convertit au passage sous la forme d'une chaîne de caractères (transtypage avec la méthode `String()`), afin d'éviter que les balises de formatage soient également affichées dans le flux du texte.

À retenir

- Le HTML peut être importé dans un document Flash et associé à une feuille de style CSS. Pour cela, nous utilisons la méthode `parseCSS()`.
- Il n'est pas nécessaire de stocker les balises dans un flux XML. Un simple fichier texte peut suffire à contenir les éléments importés. Nous utilisons alors un objet `URLVariable` pour traiter les données.

Gérer le Flash transparent

Un document Flash affiche, par défaut, une couleur d'arrière-plan. Vous pouvez indiquer au document HTML de ne pas interpréter cette donnée en vue de rendre transparent l'arrière-plan du document Flash.

L'objectif de cette modification est de permettre d'entrevoir le contenu de la page HTML généralement placée sous l'animation Flash. Cette technique est employée aussi bien pour agrémenter des contenus qui doivent se coordonner avec une image d'arrière-plan placée dans une page HTML beaucoup plus large, ou avec un motif répété à l'arrière-plan de la page HTML, par exemple. Mais elle est aussi employée dans des sites éditoriaux pour placer des annonces commerciales au premier plan sur des articles de fond, ou encore

permettre à des menus HTML / CSS de passer au devant (index-z) d'animations SWF. Sans la transparence de celui-ci cela ne serait effectivement pas possible.

Pour appliquer un fond transparent à un document SWF, dans la page HTML qui le contient, ajoutez un paramètre `wmode` et affectez la valeur `transparent` :

```
<param name="wmode" value="transparent">
```

Assurez-vous que cette option est également renseignée dans les balises `embed` et dans les paramètres du JavaScript qui contrôle éventuellement l'affichage du document Flash (voir Chapitre 15).

Attention, cette option peut perturber l'affichage des caractères dans les champs de texte dynamiques, surtout s'ils sont isolés par un masque, même en ajoutant les caractères au champ de texte dynamique, particulièrement à partir de Firefox 2 (voir aussi la section Gérer la typographie au Chapitre 15).

À retenir

- Afin de convertir l'arrière-plan d'un document SWF en fond transparent, nous définissons, dans la page HTML qui contient l'animation Flash, le paramètre `wmode` à la valeur `transparent`.

Synthèse

Dans ce chapitre, vous avez appris à établir des liaisons entre un contenu réalisé en Flash et la page HTML qui le contient, et donc, à personnaliser l'affichage d'un document Flash selon la page HTML qui s'y réfère. Vous avez également appris à organiser un historique de navigation, à personnaliser le menu contextuel de Flash et à importer du HTML dans une zone Flash. Vous savez désormais facilement intégrer vos développements dans un site web et favoriser les échanges entre contenus HTML et Flash. Des techniques complémentaires propres à l'implémentation du Flash dans le navigateur sont abordées dans le chapitre suivant.

15

La gestion de sites Full Flash

Introduction

Par gestion de sites Full Flash, nous entendons le recours exclusif à la technologie Flash pour mettre en forme un contenu web. Le contenu affiché est étendu à la surface de la fenêtre du navigateur la plus large possible, voire, sur toute la surface de l'écran.

Naturellement, même dans un site tout en Flash, une première page HTML sera nécessaire pour le contenir (généralement "index.html"). C'est, entre autres, dans cette page – à l'aide de paramètres HTML – que nous spécifions aussi la manière dont le contenu Flash peut être déployé, avec un affichage élastique ou en mode plein écran. Nous complétons ces paramétrages à l'aide d'instructions codées naturellement en ActionScript.

Réaliser un site tout en Flash peut rapidement isoler l'utilisateur dans un dispositif hermétique. Nous vous recommandons de répondre aux exigences minimales des règles d'ergonomie et d'accessibilité lorsque vous choisissez ce type de présentation. Dans ce chapitre, nous proposons une option pour basculer l'affichage en mode plein écran, à un mode normal. Ceci afin de permettre à l'utilisateur de revenir librement à d'autres tâches sur son poste de travail sans se sentir obligé de subir un mode de navigation imposé.

Nous allons voir aussi comment organiser les contenus dans Flash, afin de rendre le site structurellement élastique avec toutes les dimensions d'écran, mais surtout, comment ne pas distordre les contenus distribués lorsqu'ils sont redimensionnés.

Il est important enfin de pouvoir intégrer tout type de contenu dans un site réalisé exclusivement en Flash. Nous parcourons également, en fin de chapitre, la gestion du PDF à travers Flash ainsi que l'importation de la typographie y compris à travers des champs de texte dynamiques.

À l'issue de ce chapitre, vous serez en mesure de créer des interfaces étendues qui valorisent spatialement les contenus.

Basculer en mode plein écran et restaurer l'affichage standard

Qu'y a-t-il de plus valorisant que de voir son travail projeté en plein écran ? Lorsque l'affichage en mode plein écran est employé à bon escient, cette option peut réellement offrir une nouvelle dimension pour organiser les contenus. La surface alors disponible pour les déployer, pour un écran de résolution standard de 1 024 × 768 pixels, passe radicalement de 960 × 550 pixels, en moyenne, à 1 024 × 768 pixels. Les images, les vidéos, la 3D,

prennent une toute autre dimension. Mais, naturellement, étendre la surface des contenus suppose quelques précautions :

- La bande passante de l'utilisateur, d'abord, elle, ne change pas. Il convient donc de ne pas profiter de cette surface supplémentaire pour augmenter le poids des contenus à charger.
- L'utilisateur qui navigue ponctuellement dans votre création a en outre besoin de pouvoir quitter le mode plein écran facilement pour revenir à une autre tâche. Nous devons donc organiser la gestion de l'affichage de sorte que l'ergonomie reste viable en toute circonstance, avec un bouton de restauration de l'affichage en mode normal, par exemple.

Dans cette section, nous étudions le basculement de l'affichage en mode plein écran, ainsi que la restauration de l'affichage en mode normal.



Exemples > ch15_siteFullFlash_1 fla

Dans le document "ch15_siteFullFlash_1 fla", sur la scène principale, nous pouvons voir le MovieClip `fond_mc`, une image isolée dans un autre MovieClip, puis un bouton et un masque (voir Figures 15.1 et 15.2). À droite de la scène, le symbole `pleinEcran_btn` contient deux boutons répartis sur deux images distinctes. C'est un bouton à "basculer". Lorsque l'affichage est normal, la première image est affichée. Mais dès que l'affichage bascule en mode plein écran, c'est la seconde image qui apparaît.

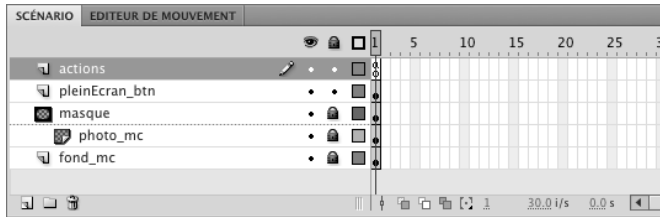
Figure 15.1

Aperçu
de la scène
principale.



Figure 15.2

Scénario de la scène principale.



Dans la fenêtre de scénario, le calque actions affiche le code suivant :

```
//----- initialisation
pleinEcran_btn.gotoAndStop(1);
pleinEcran_btn.buttonMode=true;

//----- actions
pleinEcran_btn.addEventListener(MouseEvent.CLICK,affichagePleinEcran);

function affichagePleinEcran(evt:MouseEvent) {
    if (stage.displayState==StageDisplayState.NORMAL) {
        stage.displayState=StageDisplayState.FULL_SCREEN;
        pleinEcran_btn.gotoAndStop(2);
    } else {
        stage.displayState=StageDisplayState.NORMAL;
        pleinEcran_btn.gotoAndStop(1);
    }
}
```

Le code de la page HTML qui accompagne ce document ("ch15_siteFullFlash_1.html"), intègre le Flash en utilisant les paramètres suivants :

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" codebase="http://
download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=10,0,0,0"
width="800" height="600" id="ch15_siteFullFlash_1" align="middle">
    <param name="allowScriptAccess" value="sameDomain" />
    <param name="allowFullScreen" value="true" />
    <param name="movie" value="ch15_siteFullFlash_1.swf" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="#ffffff" />
    <embed src="ch15_siteFullFlash_1.swf" quality="high" bgcolor="#ffffff"
width="800" height="600" name="ch15_siteFullFlash_1" align="middle"
allowScriptAccess="sameDomain" allowFullScreen="true" type="application/x
-shockwave-flash" pluginspage="http://www.adobe.com/go/getflash-player_fr" />
</object>
```

Dans notre exemple, le mécanisme de l’affichage plein écran s’exécute sous contrôle du navigateur. Le document Flash n’est interprété que par le lecteur Flash. Afin d’autoriser le navigateur à basculer l’affichage du Flash en mode plein écran, nous devons donc ajouter un paramètre dans la page HTML qui accueille le Flash.

Pour autoriser l’affichage en mode plein écran, dans la page HTML, au niveau des balises qui gèrent l’affichage du document Flash, vous devez ajouter le paramètre AllowFull-Screen et le définir sur true, y compris dans la balise embed :

```
<param name="allowFullScreen" value="true" />
```


Pour ce faire, vous pouvez l'ajouter manuellement à votre mise en forme, en saisissant directement dans le code. Vous pouvez aussi générer automatiquement une page HTML simple, avec ce paramètre, depuis Flash.

1. Pour publier une animation au format SWF avec une page HTML qui autorise l'affichage en mode plein écran, dans Flash, affichez les paramètres de publication, en faisant Fichier > Paramètres de publication.
2. Dans l'onglet HTML, dans le menu Modèle, sélectionnez l'option Flash seulement avec Autorisation du Plein écran (voir Figure 15.3).

Figure 15.3

Paramètres de publication HTML.



3. Puis, cliquez sur le bouton Publier, en bas de cette fenêtre, pour générer le document HTML et SWF.

Attention, le code HTML généré par Flash utilise trois portes pour inclure un objet SWF : la balise object, la balise embed et une fonction JavaScript : AC_FL_RunContent. Les navigateurs modernes se servent de ce JavaScript pour implémenter le Flash dans la page. Il faut donc s'assurer que les paramètres adéquats sont également bien présents dans cette partie du code :

```
'allowFullScreen', 'true',
```

Pour tester l'affichage en plein écran, il faudra maintenant exécuter le document SWF à partir de la page HTML, dans le navigateur, et non simplement à partir du fichier SWF lui-même.

Une fois que la page HTML dispose du paramètre AllowFullScreen, les actions du document Flash qui définissent l'affichage, peuvent être ajoutées.

La programmation d'un système d'affichage en mode plein écran, dans Flash, est astreinte à l'événement MouseEvent. Les autres événements n'autorisent pas ce type d'affichage pour

éviter naturellement les utilisations malveillantes de cette fonctionnalité. Seule une participation intentionnelle de l'utilisateur peut activer ce mode d'affichage.

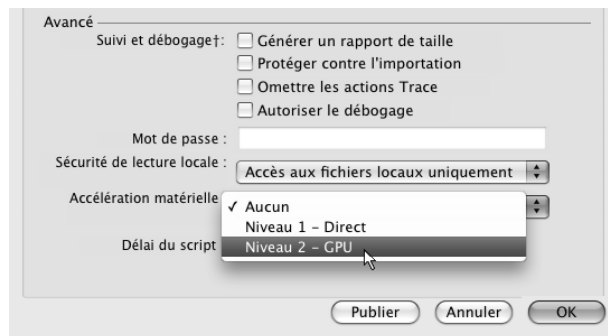


Désactivation des contrôles du clavier en plein écran. Notez que l'affichage en mode plein écran désactive les contrôles du clavier. Seul une exportation au format AIR préserve ces options, mais ce format n'est réservé que pour le développement d'applications de bureau, hors fenêtre du navigateur. Nous ne l'utilisons pas dans le cadre de la réalisation d'un site web. Pour en savoir plus sur la technologie AIR, consultez le site <http://www.adobe.com/fr/products/air/>.

Activer l'accélération matérielle pour les contenus riches. Par défaut, le contenu Flash est exécuté avec le processeur. Pour améliorer les performances d'affichage des contenus riches (vidéos, 3D, calculs graphiques permanents) affichés en plein écran par exemple, il est possible d'activer l'accélération matérielle du poste utilisateur. Pour cela, utilisez le paramètre `wmode` avec la valeur GPU, directement dans le code de la page HTML qui contient le Flash. Si la carte graphique du poste client est assez puissante, elle prendra en charge l'affichage du contenu Flash. Attention, ce mode peut modifier légèrement l'aspect des contenus. Il doit, de préférence, n'être activé que pour un seul document SWF à la fois, afin de garantir la stabilité de l'affichage dans la fenêtre du navigateur. Pour activer ce mode depuis Flash, dans les paramètres de publication, dans la partie avancée de l'onglet Flash, sélectionnez l'option Niveau 2-GPU du menu Accélération matérielle (voir Figure 15.4).

Figure 15.4

Activer l'accélération matérielle.



Dans le document Flash, le programme démarre sur l'initialisation :

```
//----- initialisation
pleinEcran_btn.gotoAndStop(1);
pleinEcran_btn.buttonMode=true;
```

Nous indiquons dans un premier temps de caler le bouton à bascule, sur la première image avec la méthode `gotoAndStop()`. Afin de laisser également apparaître la main au survol du MovieClip, nous passons la propriété `buttonMode` sur `true`.

Plus bas, un écouteur affecte une action au bouton sur un événement souris (`MouseEvent.CLICK`). Cette action active l'affichage du mode plein écran :

```
//----- actions
pleinEcran_btn.addEventListener(MouseEvent.CLICK,affichagePleinEcran);

function affichagePleinEcran(evt:MouseEvent) {
```

```
if (stage.displayState==StageDisplayState.NORMAL) {  
    stage.displayState=StageDisplayState.FULL_SCREEN;  
    pleinEcran_btn.gotoAndStop(2);  
} else {  
    stage.displayState=StageDisplayState.NORMAL;  
    pleinEcran_btn.gotoAndStop(1);  
}  
}
```

Nous pouvons basculer l'ensemble du document (stage) ou un conteneur. Il suffit de spécifier le nom de l'objet à projeter et de lui appliquer la propriété `displayState` qui désigne son état en plein écran d'affichage. Dans notre exemple, toute la scène est basculée en mode plein écran. C'est donc l'objet `stage` qui reçoit la propriété `displayState`. Nous définissons, pour cet objet, le type d'affichage à l'aide de la valeur `stageDisplayState.FULL_SCREEN` :

```
stage.displayState=StageDisplayState.FULL_SCREEN;
```

Nous avons ajouté une condition qui vérifie l'état d'affichage. Si l'affichage est normal, nous basculons en plein écran. Si, inversement, l'état d'affichage est déjà en plein écran, nous spécifions l'action inverse, de restaurer l'affichage :

```
stage.displayState=StageDisplayState.NORMAL;
```

Pour basculer d'une icône à une autre, nous ajoutons aussi une action `gotoAndStop()` qui cible l'image correspondant à l'action disponible pour chaque mode d'affichage. Cette condition permet d'appliquer une action à bascule sur un même et unique bouton. Mais vous pouvez bien entendu séparer les instructions pour les isoler sur des objets distincts.

En projetant la page HTML dans le navigateur, et en activant le bouton `pleinEcran_btn`, le site apparaît en plein écran. En cliquant à nouveau, il revient à un affichage normal (voir Figure 15.5).

Figure 15.5

Affichage en plein écran.





Quitter le mode Plein écran avec Echap. Indépendamment des contrôles de bouton développés manuellement, Flash active, par défaut, une option de restauration de l’affichage si vous appuyez sur la touche Echap du clavier. Un message apparaît pour prévenir de cette option dès le passage en mode plein écran. Il n’y a pas de contrôle pour modifier ce message afin de garantir à tout utilisateur une sortie possible, quel que soit le type de développement ajouté. Mais nous pouvons détecter l’action sur le bouton esc... pour repositionner notre bouton bascule à son état normal :

```
stage.addEventListener(FullScreenEvent.FULL_SCREEN, ecoute);
function ecoute(evt:FullScreenEvent){
    if (! event.fullScreen) {
        pleinEcran_btn.gotoAndStop(1);
    }
}
```

À retenir

- Le mode d’affichage en plein écran se définit par la propriété `stageDisplayState`.
- Pour activer l’affichage en mode plein écran, la page HTML qui contient le Flash doit également contenir le paramètre `AllowFullScreen` et le passer sur `true`.
- Les actions de clavier sont inactives en mode plein écran.
- Le mode plein écran n’est disponible qu’à travers une action souris.

Interface élastique flottante

Le principe d’une interface élastique est d’appliquer, pendant le redimensionnement de la fenêtre du navigateur, les proportions de cette fenêtre au document Flash, dans le but de remplir systématiquement la surface de la fenêtre quelles qu’en soient les dimensions.

Cela se construit initialement de manière assez simple puisqu’il suffit de renseigner une largeur et une hauteur de 100 % dans les propriétés de dimensions du Flash du document HTML. Mais si l’on ne veut pas que les contenus soient déformés, et qu’ils occupent, malgré tout, l’intégralité de la surface de la fenêtre, seules des instructions en ActionScript permettent de redistribuer les contenus en contrôlant leurs dimensions, leurs proportions et leurs positions en X et Y.

Dans cette section, nous allons voir comment placer des `MovieClip` dans la scène de sorte qu’elle s’étende, quelle que soit la taille de la fenêtre, mais sans jamais distordre les contenus ni laisser de vide dans le document.



Exemples > `ch15_siteFullFlash_2 fla`

Dans le document "`ch15_siteFullFlash_2 fla`", sur la scène, nous distinguons à l’arrière-plan une image de 800×600 pixels placée dans un `MovieClip` `contenu_mc` (voir Figure 15.6). Au-dessous, à droite, apparaît un `MovieClip` `texteDroite_mc`, qui contient le titre du livre. À gauche, un autre `MovieClip` affiche le logo de l’éditeur. Le filet est intégré dans le clip du

bandeau bleu, calé, lui, en bas de la fenêtre. Tout en haut du document, apparaît aussi un symbole nommé `carte_mc` qui représente le département du Vaucluse.

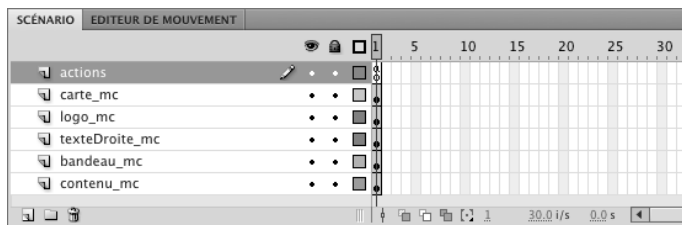
Figure 15.6

Aperçu du document.



Figure 15.7

Scénario de la scène principale.



Dans la fenêtre Actions (voir Figure 15.7), nous pouvons lire le code suivant :

```
//----- initialisation
stage.scaleMode=StageScaleMode.NO_SCALE;
stage.align=StageAlign.TOP_LEFT;

//----- actions
stage.addEventListener(Event.RESIZE, quandResizeActif);
function quandResizeActif(event:Event) {
    modifierAffichage();
}

function modifierAffichage() {
    //
    logo_mc.x=20;
    logo_mc.y=stage.stageHeight-35;
    //
    texteDroite_mc.x=stage.stageWidth-20;
    texteDroite_mc.y=stage.stageHeight-35;
```

```
//
bandeau_mc.x=stage.stageWidth/2;
bandeau_mc.y=stage.stageHeight-(bandeau_mc.height/2);
bandeau_mc.width=stage.stageWidth;
//
contenu_mc.x=0;
contenu_mc.y=0;
contenu_mc.width=stage.stageWidth;
contenu_mc.height=stage.stageHeight;
if (contenu_mc.scaleX<=contenu_mc.scaleY) {
    contenu_mc.scaleX=contenu_mc.scaleY;
} else {
    contenu_mc.scaleY=contenu_mc.scaleX;
}
}
modifierAffichage();
```

La scène, en ActionScript 3, possède des propriétés de largeur (`stageWidth`) et de hauteur (`stageHeight`), que nous avons déjà rencontrées dans les chapitres précédents. Grâce à ces propriétés, nous connaissons à tout moment les dimensions du document. Nous pouvons donc récupérer ces valeurs pour redéfinir, dynamiquement, l'emplacement des objets par rapport à la taille du document.

Nous pouvons, par exemple, jouer sur les positions X et Y, pour caler systématiquement un logo en bas et à gauche. Nous pouvons, aussi, dans le cas où certains éléments doivent être redimensionnés, modifier leurs propriétés `width` et `height` en proportions des largeur et hauteur de la scène.

Dans cet exemple, nous plaçons les objets tantôt selon la largeur du document, tantôt selon sa hauteur. Pour deux d'entre eux, l'image de fond ainsi que le bandeau du bas, nous redéfinissons leurs dimensions en proportion des dimensions de la scène.

Dans la première partie du code, nous définissons le comportement de l'affichage de la scène par défaut :

```
//----- initialisation
stage.scaleMode=StageScaleMode.NO_SCALE;
```

La propriété `scaleMode` permet d'indiquer si les contenus sont déformés ou non lorsque la taille de la fenêtre du document est modifiée. La valeur `stageScaleMode.NO_SCALE` indique qu'aucune déformation n'est appliquée aux contenus.



Modes d'affichage de la scène. Il existe quatre modes d'affichage de la scène courante : `NO_SCALE`, `EXACT_FIT`, `NO_BORDER` et `SHOW_ALL`. Elles définissent le comportement des objets dans la scène, lors du redimensionnement du document Flash dans la fenêtre de navigateur. Les modifications de propriété `width` et `height`, ajoutées éventuellement en ActionScript sur des objets demeurent prioritaires sur ces options-ci.

- `NO_SCALE` empêche les modifications d'échelle des contenus même si les dimensions de la fenêtre sont modifiées.
- `EXACT_FIT` autorise la distorsion des contenus selon l'exakte proportion des dimensions de la fenêtre.

- `NO_BORDER` préserve les proportions de la scène quelle que soit la taille de la fenêtre. La scène est agrandie et recadrée si nécessaire. Le contenu occupe toujours intégralement la fenêtre.
- `SHOW_ALL` préserve également les proportions de la scène et l'agrandit si nécessaire, mais ne recadre jamais le contenu. Des marges sont ajoutées autour de la scène si les proportions de la fenêtre sont différentes de celles de la scène.

La deuxième instruction emploie la propriété `align` qui définit l'alignement de la scène, lorsque la fenêtre est redimensionnée :

```
stage.align=StageAlign.TOP_LEFT;
```

Dans notre exemple, la valeur `TOP_LEFT` désigne un calage en haut et à gauche. Ainsi, si nous n'avions pas modifié dynamiquement le positionnement de tous les objets, ils resteraient à leur place actuelle et le redimensionnement de la fenêtre impliquerait l'ajout de marges à droite et en bas du document.

Le langage

Propriétés d'alignement de la scène

- `BOTTOM`. La scène est alignée sur le bas.
- `BOTTOM_LEFT`. La scène est alignée sur le coin inférieur gauche.
- `BOTTOM_RIGHT`. La scène est alignée sur le coin inférieur droit.
- `LEFT`. La scène est alignée sur la gauche.
- `RIGHT`. La scène est alignée sur la droite.
- `TOP`. La scène est alignée sur le haut.
- `TOP_LEFT`. La scène est alignée sur le coin supérieur gauche.
- `TOP_RIGHT`. La scène est alignée sur le coin supérieur droit.

Nous détectons ensuite le redimensionnement de la fenêtre en ajoutant un écouteur à la scène (`stage`). La propriété `RESIZE` permet d'exécuter la fonction durant chaque redimensionnement de fenêtre et uniquement là :

```
//----- actions
stage.addEventListener(Event.RESIZE, quandResizeActif);
function quandResizeActif(event:Event) {
    modifierAffichage();
}
```

La fonction invoquée appelle une autre fonction, autonome :

```
modifierAffichage();
```

Nous avons choisi d'isoler les actions dans une deuxième fonction afin de permettre de les appeler depuis différents emplacements et selon différents types d'événements. En l'occurrence, nous souhaitons exécuter les instructions pendant le redimensionnement, mais aussi dès l'ouverture du document. Si nous ne redéfinissions pas le positionnement dès l'ouverture de l'animation, nous risquerions en effet d'observer un saut ou un décalage entre la position courante et celle qui est redéfinie au premier redimensionnement.

La fonction appelée redéfinit les propriétés x, y, width et height des différents contenus selon les largeurs et hauteur de la scène :

```
function modifierAffichage() {
    //
    logo_mc.x=20;
    logo_mc.y=stage.stageHeight-35;
    //
    texteDroite_mc.x=stage.stageWidth-20;
    texteDroite_mc.y=stage.stageHeight-35;
    //
    bandeau_mc.x=stage.stageWidth/2;
    bandeau_mc.y=stage.stageHeight-(bandeau_mc.height/2);
    bandeau_mc.width=stage.stageWidth;
    //
    contenu_mc.x=0;
    contenu_mc.y=0;
    contenu_mc.width=stage.stageWidth;
    contenu_mc.height=stage.stageHeight;
    if (contenu_mc.scaleX<=contenu_mc.scaleY) {
        contenu_mc.scaleX=contenu_mc.scaleY;
    } else {
        contenu_mc.scaleY=contenu_mc.scaleX;
    }
}
```

Les premières lignes récupèrent simplement les valeurs stage.stageWidth et stage.stageHeight. Des marges sont toutefois ajoutées ou soustraites en fonction de la position réellement souhaitée pour chaque objet. Par exemple, le symbole logo_mc est placé systématiquement à 20 pixels du bord gauche de la fenêtre, mais toujours calé en bas à 35 pixels du bord (stage.stageHeight-35 équivaut à dire : la hauteur moins 35 pixels).

Lorsque vous positionnez les objets, pensez à tenir compte de leur point d'alignement (centre) et de leur hauteur ou largeur. Ceci est déterminant pour caler parfaitement les éléments dans la scène.

Le MovieClip bandeau_mc est positionné en Y selon la même hauteur de fenêtre. Mais il est également redimensionné en largeur (width) selon la largeur effective de la fenêtre. Le symbole occupe donc toute la largeur de la scène quelle qu'en soit la taille. Nous avons bien entendu pris soin d'y placer des graphismes qui supportent cette distorsion.

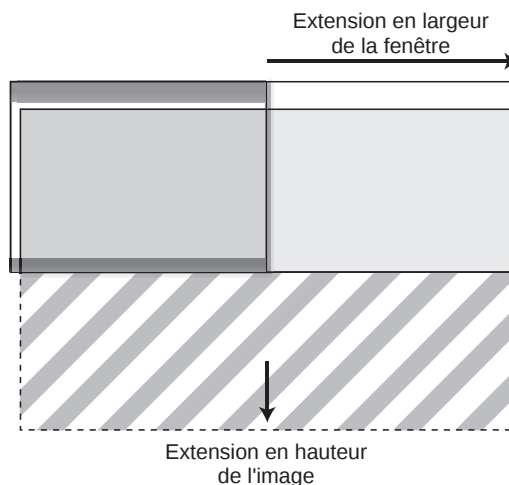
La dernière partie de la fonction affecte également les positions X et Y, ainsi que les dimensions width et height de l'objet contenu_mc. Mais, nous ajoutons ici une condition :

```
contenu_mc.x=0;
contenu_mc.y=0;
contenu_mc.width=stage.stageWidth;
contenu_mc.height=stage.stageHeight;
if (contenu_mc.scaleX<=contenu_mc.scaleY) {
    contenu_mc.scaleX=contenu_mc.scaleY;
} else {
    contenu_mc.scaleY=contenu_mc.scaleX;
}
```


Lorsque la fenêtre est étirée en largeur, pour remplir la surface de la fenêtre, nous devons étirer le contenu également en largeur. Mais, afin de préserver ses proportions, nous devons l'étirer aussi dans le sens de la hauteur, hors champ (voir Figure 15.8).

Figure 15.8

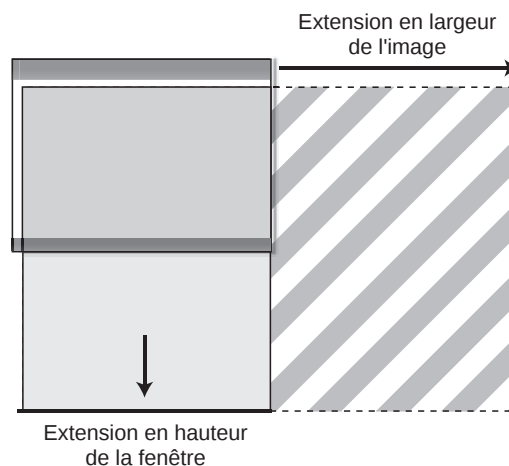
Étirement
du contenu
à la verticale.



Inversement, lorsque la fenêtre est étirée en hauteur, nous devons agrandir le contenu en hauteur et, indirectement, en largeur, hors champ (voir Figure 15.9).

Figure 15.9

Étirement
du contenu à
l'horizontal.



Il en résulte que l'image remplit la surface de la fenêtre en toutes circonstances.

Mais, nous ne pouvons à la fois dire que la largeur du contenu correspond à la largeur de la fenêtre et que la hauteur de ce même contenu équivaut également la hauteur de la fenêtre. Car il suffit que la fenêtre n'affiche pas les mêmes proportions que celles du contenu pour que les instructions entrent en conflit. Pour résoudre ce dilemme, nous devons spécifier une action qui active l'un ou l'autre des redimensionnements, en fonction de la plus grande

longueur observée. Si c'est la largeur qui est plus grande que la hauteur, alors, nous étirons l'image en hauteur. Si, en revanche, c'est la hauteur de la fenêtre qui est plus importante, alors, nous étirons l'image dans sa largeur. Comme, dans les deux cas, l'image conserve ses proportions, elle remplira toujours la surface de l'écran.

Nous précisons alors, dans notre fonction, la condition suivante :

```
if (contenu_mc.scaleX<=contenu_mc.scaleY) {  
    contenu_mc.scaleX=contenu_mc.scaleY;  
} else {  
    contenu_mc.scaleY=contenu_mc.scaleX;  
}
```

Si l'échelle de redimensionnement du contenu en largeur (scaleX) est inférieure à son redimensionnement en hauteur (scaleY) (donc, si la fenêtre est plus haute que large), alors, l'échelle en largeur (scaleX) prend pour valeur l'échelle de la hauteur (scaleY) :

```
if (contenu_mc.scaleX<=contenu_mc.scaleY) {  
    contenu_mc.scaleX=contenu_mc.scaleY;  
}
```

Inversement, si c'est la largeur qui est plus grande que la hauteur, dans ce cas, nous appliquons l'échelle de la hauteur (scaleY) sur l'échelle de la largeur (scaleX) :

```
else {  
    contenu_mc.scaleY=contenu_mc.scaleX;  
}
```

Cette technique permet de redimensionner la fenêtre dans des proportions libres, différentes de celles de la scène, et d'élargir quand même le contenu, mais sans jamais y appliquer de distorsion. Seule une transformation homothétique est appliquée.

Pour assurer une remise à l'échelle de qualité correcte, pensez également à appliquer, aux contenus, un lissage. Pour en savoir plus sur cette technique, reportez-vous au chapitre consacré aux API d'affichage et de filtres. Dans ce document, nous avons simplement appliqué un lissage depuis la fenêtre Bibliothèque.



Principe de flottement façon styles CSS. En déclinant le principe de redimensionnement, ici évoqué, et celui du repositionnement, développé dans la section des menus déroulants, vous pouvez aussi concevoir une interface dans laquelle les objets flottent de manière relative, les uns par rapport aux autres, et se redimensionnent selon la taille effective de la fenêtre du navigateur, à la manière d'une page HTML, par exemple.

À retenir

- Pour placer et redimensionner des objets en fonction de la taille de la fenêtre, nous utilisons les propriétés `stageWidth` et `stageHeight` de la scène.
- Afin d'appliquer une mise à l'échelle homothétique, nous ajoutons une condition qui détermine le redimensionnement en fonction de la largeur la plus grande. Cette largeur est ensuite utilisée pour définir le coefficient de redimensionnement.

Gérer le PDF

Une interface tout en Flash implique la nécessité d'y importer différents types de médias, afin, tout simplement, de ne pas avoir à sortir de cet espace pour y accéder. Typiquement, cela pose problème lorsqu'un lien, dans Flash, cible un document PDF. Dans ce contexte, le format PDF, qui requiert l'ouverture d'Acrobat, oblige l'utilisateur à sortir du site tout en Flash pour pouvoir lire un contenu.

Dans cette section, bien que cela soit structurellement impossible, nous abordons les contournements qui permettent d'intégrer malgré tout un document PDF dans Flash, afin de permettre à l'utilisateur de rester dans son univers développé tout en Flash.



Différence structurelle entre Flash et Acrobat. Flash compile des données à la publication du document. Acrobat, à l'inverse, ne gère qu'un amoncellement de scripts assemblés dans un certain ordre, sans les compiler. Acrobat agit un peu à la manière d'une page HTML qui gère des fonctions JavaScript et de médias zippés. La structure des documents étant très différente, l'implémentation du format non compilé dans le format compilé est peu concevable. L'inverse, en revanche, se fait facilement. Vous pouvez parfaitement placer un contenu SWF dans un document Acrobat.

Il n'est pas possible d'afficher dynamiquement un PDF dans l'interface de Flash. Mais, il est possible d'importer un PDF physiquement, dans la scène, avant publication. Il est également possible de générer dynamiquement des fichiers au format PDF à partir d'un document Flash. Vous pouvez aussi appeler un fichier PDF à travers un hyperlien en utilisant la méthode `navigateToURL`, abordée au Chapitre 14. Quelques services en ligne proposent aussi de vous fournir dans un format compatible Flash, un PDF recompilé.

Pour importer physiquement un PDF dans Flash, vous devez d'abord le convertir dans un autre format *via* InDesign ou Illustrator, par exemple, les formats compatibles avec Flash sont l'EPS, le format AI et le XFL. Naturellement, les formats d'images sont également compatibles, mais alourdissent considérablement le poids des contenus et la gestion des documents.

1. Depuis InDesign, faites Fichier > Exporter (Ctrl+E pour Windows ou Cmd+E pour Macintosh).
2. Puis, dans la fenêtre d'exportation, dans le menu Format, choisissez "Adobe Flash CS4 prof (XFL)". Validez.
3. En validant, une boîte de réglage apparaît (voir Figure 15.10).
4. Spécifiez un texte InDesign en texte Flash, afin de préserver l'édition du document. Puis, validez.
5. Depuis Flash, faites ensuite Fichier > Ouvrir.
6. Sélectionnez le document XFL et confirmez l'importation.

Le document est importé dans la scène et vous pouvez l'animer. Le procédé est similaire avec Illustrator.

Vous pouvez aussi convertir un PDF en livre à feuillets depuis InDesign. Reportez-vous également au Chapitre 9 pour en savoir plus à ce sujet.

Figure 15.10

Scénario de la scène principale.



Convertir un PDF en SWF avec FlashPaper. Il est possible de générer un document Acrobat PDF en fichier Flash SWF, à la volée. L'utilitaire FlashPaper, développé par la société Macromedia, et compatible uniquement avec Windows, offre cette possibilité. Une version d'évaluation de ce programme est disponible à l'adresse suivante : <http://www.adobe.com/cfusion/tdrc/index.cfm?product=flashpaper>.

Création dynamique de fichiers PDF. Pour générer dynamiquement un PDF, nous pouvons utiliser la classe `ByteArray`. Reportez-vous au Chapitre 5 du livre de Thibault Imbert, éditions Pearson, *Pratique d'ActionScript 3*, pour en savoir plus à ce sujet.

À retenir

- Il est possible d'importer un document PDF au sein d'une animation Flash. Pour cela, nous devons au préalable le convertir en dur depuis une application graphique ou dynamiquement.

Gérer la typographie

La gestion de la typographie, dans une page web, soulève toujours de nombreuses interrogations. Usuellement, une page web dépend des polices présentes sur le système de l'utilisateur pour afficher un texte. Dans une page HTML, nous spécifions alors la police à employer. Si celle-ci est absente de la configuration de l'utilisateur, c'est la police activée par défaut, par le navigateur, qui prend le relais (généralement Times).

Dans Flash, c'est différent. Flash propose trois modes de traitement pour le texte : les textes statiques, les textes dynamiques et les textes de saisie. Nous ne comptons pas les textes vectorisés sous forme de graphisme ni les textes véhiculés sous la forme d'images.



Vectoriser un texte dans Flash. Pour vectoriser un texte dans Flash, faites deux fois de suite Ctrl+B (Windows) ou Cmd+B (Mac). Cette commande casse l'enveloppe courante, qu'elle soit un bloc de texte ou un symbole. Pour le texte, la première action sépare les lettres, la seconde les vectorise. Petit rappel mnémotechnique : pensez à B comme *Break* qui signifie casser en anglais. Une fois le texte vectorisé, pensez également à le regrouper afin d'éviter tout risque de morcellement si vous

devez ensuite le manipuler. Pour le regrouper, faites Ctrl+G (Windows) ou Cmd+G (Mac). Attention, un texte vectorisé n'est plus éditable. Mais la forme graphique devient modifiable avec les outils de dessin (Plume, pot de peinture, outils de sélection).

Les textes statiques

Les champs de textes statiques intègrent la forme de contour vectorielle des caractères, à la publication du document. Les caractères sont donc pris en charge nativement.



Affichage des polices dans Flash. Attention, toutes les polices affichées correctement dans le document Flash ne sont pas nécessairement intégrées à la publication. Pour vérifier qu'une police est intégrée, vérifiez que l'option de menu Affichage > Mode Aperçu > Texte anti-aliasé, est bien cochée. Si une police s'avère non intégrable, elle apparaît alors avec un crénelage dans le document. Vous pouvez ainsi en choisir une autre depuis le menu Famille de caractère, de l'inspecteur de propriétés.

Les documents Flash peuvent utiliser des polices Type 1 PostScript®, TrueType® et bitmap (uniquement sur Macintosh) de taille maximum de 255 points. Les textes sont Unicode depuis la version du lecteur Flash 7.

Les textes dynamiques

Les textes dynamiques et de saisie, en revanche, utilisent les polices de périphérique (_sans, _serif, ou _typewriter). Ces polices sont disponibles en tête de liste dans la liste des polices de l'inspecteur de propriétés. La police _sans correspond à de l'Arial ou de l'Helvetica. La police _serif, à un Times. La police _typewriter à un Courier.

Si vous souhaitez ajouter une typographique spécifique à un texte dynamique, cela reste possible. Il suffit d'en ajouter les caractères nécessaires, sur chaque champ concerné, depuis l'inspecteur de propriétés.

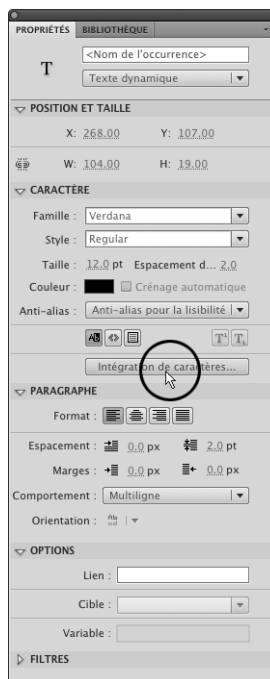
1. Pour ajouter une police à un champ de texte dynamique, sélectionnez le champ de texte avec l'outil de sélection (flèche noire).
2. Dans l'Inspecteur de propriétés, cliquez sur Intégration de caractères (voir Figure 15.11).
3. Une boîte de dialogue s'ouvre.
4. Dans la liste qui apparaît, sélectionnez uniquement les caractères utiles pour optimiser autant que possible le poids du document (voir Figure 15.12). Au besoin, ajoutez manuellement les caractères manquants, dans le champ Inclure ces caractères.

Lorsque vous ajoutez des caractères manuellement, relevez que chaque caractère ajouté augmente d'autant le poids du fichier. Éviter, par conséquent, de recourir à un nombre trop varié de polices de caractère dans un même document.

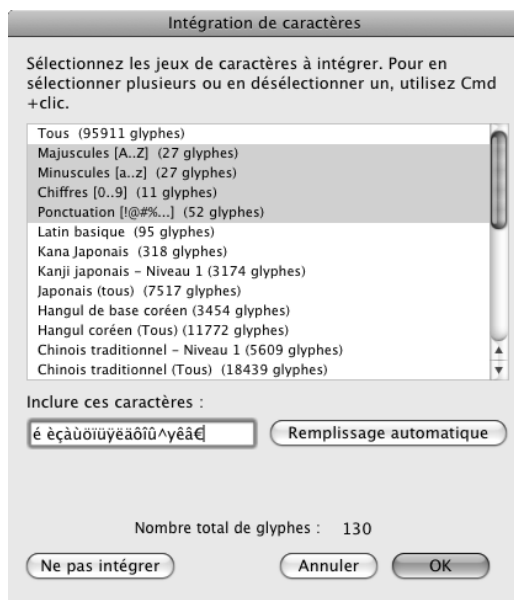
Lorsque vous saisissez les caractères à ajouter, pensez à ajouter aussi l'espace, les sigles monétaires et les caractères accentués ou spéciaux. Les listes de caractères par défaut étant conçues initialement pour un public anglo-saxon, il convient d'ajouter ceux qui ne figurent pas dans cet alphabet, comme les caractères é, è, ç, à, ô, î, û, ê, â, ö, ï, ü, ë, ä, ı et espace.

Figure 15.11

Inspecteur
de propriétés.

**Figure 15.12**

Intégration
de caractères.



Dans la Figure 15.12, nous avons pris soin de ne sélectionner que les listes de caractères Minuscules, Majuscules, Chiffres et Ponctuations, puis de saisir, à la main, les caractères spéciaux manquants, directement dans le champ Inclure les caractères. La liste "Tous" intègre

absolument tous les caractères, vous augmenteriez considérablement le temps de publication et le poids de votre document, en la sélectionnant. Il est en outre préférable de restreindre le choix aux caractères utiles uniquement et d'y ajouter, si nécessaire, les caractères manquants, dans le champ Inclure les caractères, prévu à cet effet.

Attention, l'option Remplissage automatique n'enregistre que les caractères affichés dans le champ actif.



Les polices et les masques. Si vous utilisez des champs de textes dynamiques ou de saisie, avec des masques, vérifiez toujours la saisie ou l'affichage sur des postes utilisateurs différents. Selon la configuration du système, certains caractères peuvent ne pas apparaître. Dans ce cas, pensez déjà à ajouter les caractères manquants. Notez que le paramètre `wmode` utilisé pour un affichage transparent du document Flash (vu précédemment) peut aussi neutraliser certains caractères à la saisie (arobase), si un masque est utilisé, avec d'anciennes versions du lecteur Flash. Dans ce cas, évitez d'utiliser conjointement les masques et les champs de textes dynamiques.

Les bibliothèques partagées de symboles et de typographies

Le principe de la bibliothèque partagée est de centraliser, sous la forme d'un fichier SWF ou SWC, un ou plusieurs objets de la bibliothèque d'un document Flash. Cette technique permet d'optimiser la gestion d'objets récurrents sans avoir à les stocker à l'intérieur de chaque document SWF qui doit les utiliser. Sachant combien le poids d'une typographie peut être compromettant dans l'élaboration d'un site web, et combien son utilisation affecte, pourtant, les identités graphiques, rassembler une typographie dans une bibliothèque partagée peut nous affranchir de quelques contraintes.



Exemples > `ch15_siteFullFlash_3 fla`

Exemples > `ch15_siteFullFlash_3b fla`

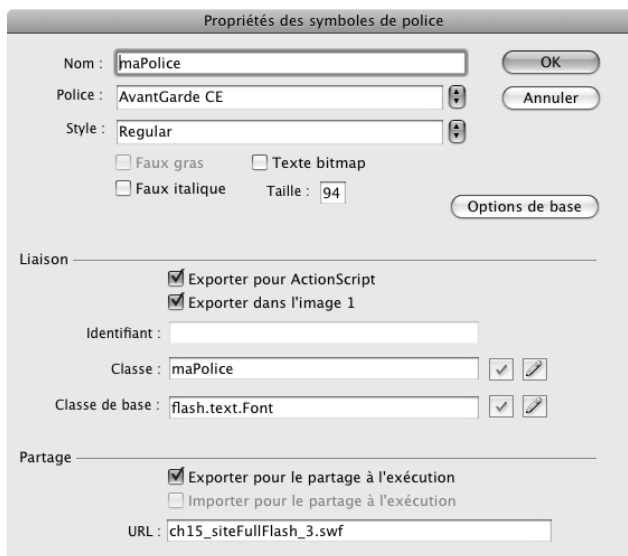
Dans cette section, nous présentons comment exporter une police à partir d'un document Flash et l'intégrer dynamiquement à tout autre document SWF. Les deux fichiers d'exemple "`ch15_siteFullFlash_3 fla`" et "`ch15_siteFullFlash_3b fla`" reprennent cette procédure.

Pour partager une typographie, dans le menu contextuel de la fenêtre de bibliothèque d'un document Flash, choisissez Nouvelle police. Une fenêtre de dialogue apparaît (voir Figures 15.13 à 15.15).

1. Dans le champ Nom, inscrivez le nom que vous souhaitez attribuer à votre police partagée.
2. Dans le menu Police, sélectionnez la typographie de votre choix. Ajoutez éventuellement quelques options comme l'italique, le gras, si ces propriétés sont disponibles par exemple.
3. Dans la partie centrale, activez l'option Exporter pour ActionScript si vous souhaitez pouvoir disposer de la typographie en programmation pour des objets TextField. Et attribuez dans ce cas un nom de classe à la police.

Figure 15.13

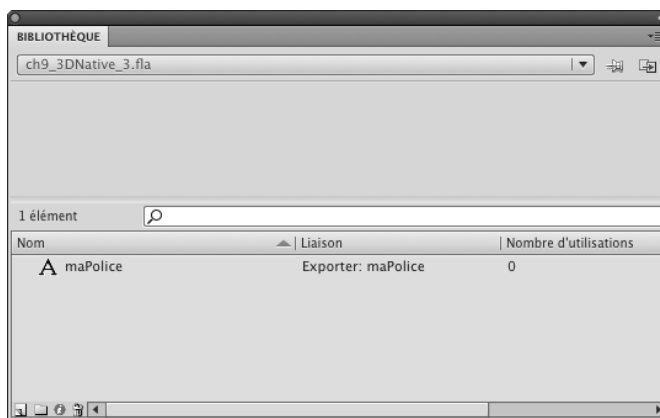
Ajout d'une police partagée à la bibliothèque.



4. Dans la partie inférieure, activez l'option Exporter pour le partage à l'exécution. Dans le champ URL, maintenant actif, inscrivez le nom de publication du fichier SWF de ce document Flash. Puis validez. La fenêtre de bibliothèque affiche la nouvelle police. La colonne Liaison indique que l'objet est prêt pour l'exportation (voir Figure 15.14).

Figure 15.14

Affichage de la police partagée dans la bibliothèque.



5. Procédez de même pour des symboles, en créant par exemple un movieClip. Puis, dans la fenêtre de bibliothèque, affichez les propriétés contextuelles du movieClip en faisant un clic-droit > Propriété, sur le symbole. La fenêtre de propriétés du symbole apparaît (voir Figure 15.15).
6. Activez les options d'exportation et validez.

Figure 15.15

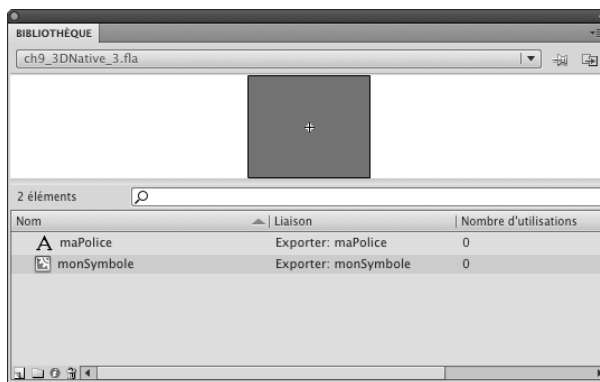
Ajout d'un symbole partagé à la bibliothèque.



La fenêtre de bibliothèque affiche également le MovieClip et son statut d'objet pour l'exportation (voir Figure 15.16).

Figure 15.16

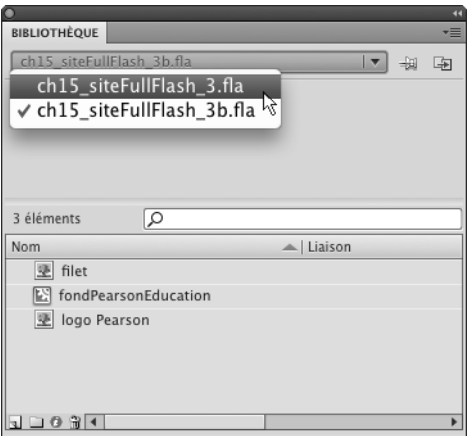
Affichage du symbole partagé dans la bibliothèque.



7. Sans nécessairement utiliser les objets dans la scène de ce document, publiez-le au format SWF en faisant Ctrl+Entrée (Windows) ou Cmd+Entrée (Macintosh).
8. Puis, dans un nouveau document Flash que vous aurez préalablement enregistré, importez les éléments partagés par glisser-déposé de la bibliothèque du document exportateur vers la scène du document importateur (voir Figure 15.17).
9. Pour accéder à la bibliothèque d'un autre document, depuis le sommet de la fenêtre de bibliothèque du document actif, sélectionnez dans la liste déroulante, le nom du document contenant les objets à importer.

Figure 15.17

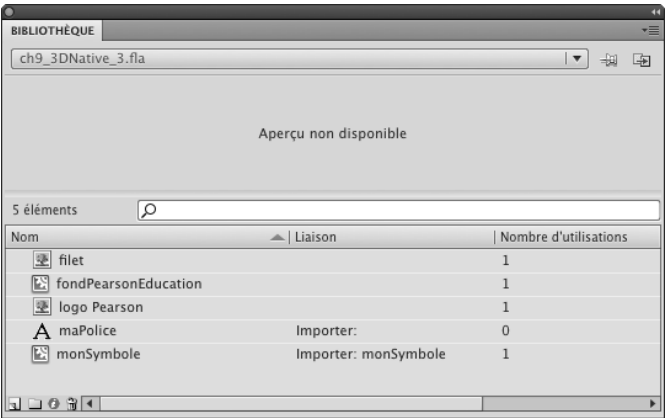
Importation dynamique des objets partagés dans un autre document..



La fenêtre de bibliothèque du nouveau document affiche le statut des objets importés une fois ceux-ci ajoutés (voir Figure 15.18).

Figure 15.18

Affichage des objets importés dans la bibliothèque du nouveau document.



10. Dans la scène du nouveau document, placez une occurrence du MovieClip importé et ajoutez un champ de texte en y inscrivant quelques mots. Pour appliquer la police importée au champ de texte, depuis l'inspecteur de propriétés du texte, dans le menu Famille de caractère, sélectionnez le nom que vous avez attribué à votre police, désormais disponible. Le champ de texte affiche la police sélectionnée. Dans le document "ch15_siteFullFlash_3b fla", nous utilisons les objets partagés du document "ch15_siteFullFlash_3 fla" (voir Figure 15.19). Publiez le nouveau document au format SWF.
11. Puis, revenez dans le document initial pour modifier la forme intérieure du MovieClip original et la police. Pour la modifier, depuis les propriétés de la police, disponibles à partir de la fenêtre de bibliothèque, sélectionnez une autre famille de caractères et validez votre modification. Publiez à nouveau ce document.

Figure 15.19

Utilisation
des objets
partagés dans
le document..



12. Sans republier le nouveau fichier Flash qui importe les objets partagés, exécutez directement son format déjà publié en SWF. Les objets sont instantanément mis à jour..



Le nouveau moteur de texte Text Layout Framework. Adobe prévoit l'implémentation d'un nouveau moteur de gestion de texte pour les versions à venir de Flash. Il permet de gérer le flottement du texte sur plusieurs colonnes, dynamiquement, la chasse, l'interlignage, la sélection, comme dans un véritable logiciel de mise en pages. Ce moteur est actuellement disponible en version bêta. Vous pouvez le tester à l'adresse suivante : <http://labs.adobe.com/technologies/textlayout/>.

À retenir

- Il est possible de partager des symboles et des polices de caractère pour plusieurs documents Flash, simultanément. Pour cela, nous activons l'option Exporter pour le partage à l'exécution, depuis les propriétés des objets disponibles à partir de la bibliothèque et importons cette bibliothèque dans d'autres documents Flash.

Importer Flash AS1/AS2 dans Flash AS3

Les documents codés en AS1 ou AS2 utilisent le même moteur (AVM1). Le langage ActionScript 3 est, lui, structurellement différent des précédentes versions et utilise un nouveau moteur (AVM2). Les documents Flash basés sur AS3 disposent donc d'une architecture différente des anciens documents Flash. Si nous pouvons facilement importer des contenus AS1 ou AS2 animés dans un document AS3, à partir d'un simple chargeur, nous ne pouvons pas, en revanche, y apporter de l'interactivité. Pour cela, nous devons ouvrir une connexion entre les deux formats de fichier, par nature dissociés, à l'aide de la classe LocalConnexion.

Pour comprendre comment utiliser cette classe, nous devons admettre que le fichier AS1/AS2 d'une part et AS3 d'autre part, sont hermétiques l'un envers l'autre, et que seul un objet `LocalConnexion` permet d'ouvrir une porte de chaque côté des deux animations pour transmettre des requêtes d'AS3 vers AS2.

Il devient alors facile d'invoquer une fonction AS2, depuis une instruction placée dans un document AS3, par simple identification du nom de la connexion et de la fonction, définies dans le document AS2.

Le fichier AS3 émetteur est appelé fichier d'envoi. Il contient la méthode à appeler. Le fichier d'envoi doit contenir un objet `LocalConnection` et un appel de la méthode `send()`. L'autre fichier, le fichier AS1 ou AS2, est dit de réception. Il s'agit de celui qui appelle la méthode. Le fichier de réception doit contenir un autre objet `LocalConnection` et un appel de la méthode `connect()`.

Dans cette section, nous présentons un document AS3 qui contient un bouton. Ce bouton affecte les propriétés d'un symbole de type `MovieClip`, contenu dans un fichier AS2 importé.

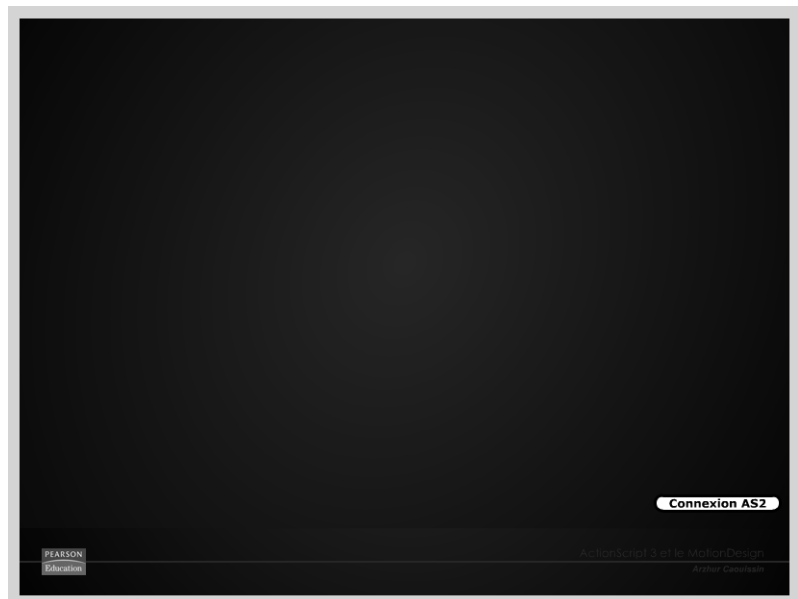


Exemples > ch15_siteFullFlash_4_AS3 fla
Exemples > ch15_siteFullFlash_4b_AS2 fla

Le document "ch15_siteFullFlash_4_AS3 fla" affiche une scène exportée pour Flash 10 et ActionScript 3.0. Sur cette scène figure un bouton nommé `connexion_btn` (voir Figure 15.20).

Figure 15.20

Aperçu du document AS3.



Le fichier "ch15_siteFullFlash_4b_AS2.fla", affiche quant à lui une scène exportée pour ActionScript 2.0 et le lecteur Flash 8. Sur cette scène, figure un MovieClip nommé `symbole_mc` (voir Figure 15.21).

Figure 15.21

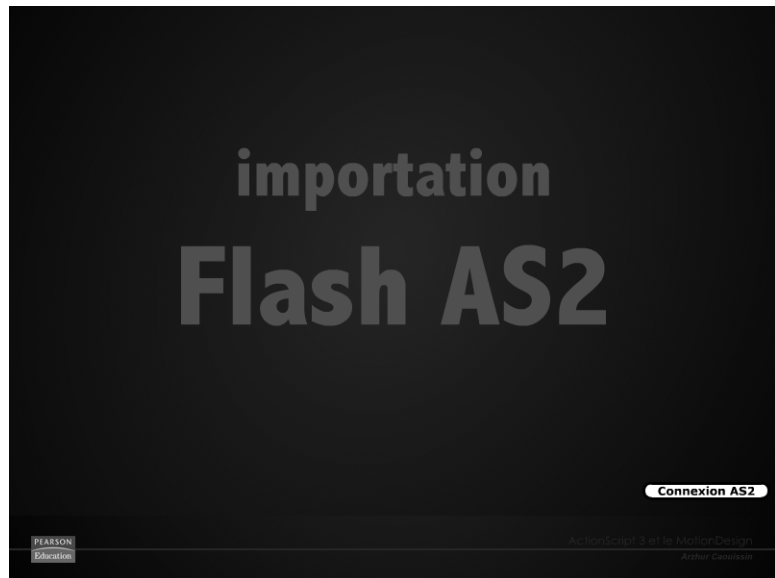
Aperçu du document AS2.



En publiant le document AS3, le fichier AS2 est d'abord importé en premier plan. En cliquant sur le bouton `connexion_btn`, du document AS3, le symbole du fichier AS2 subit une rotation de 20° (voir Figure 15.22).

Figure 15.22

Interaction AS3 vers AS2.



Dans la fenêtre Actions du document AS2 ("ch15_siteFullFlash_4b_AS2.fla"), nous pouvons lire le code suivant :

```
// récepteur  
var recepneur:LocalConnection = new LocalConnection();
```

```
// active la connexion
recepteur.connect("maConnexionAS2");

// définition de la fonction AS2 activée depuis le document AS3
recepteur.monActionProgrammeeAS2=function () {
    // actions AS2
    symbole_mc._rotation+=20;
}
```

Dans le code du document AS3 ("ch15_siteFullFlash_4_AS3 fla"), nous lisons le programme suivant :

```
// chargeur
var chargeur:Loader = new Loader();
var chemin:URLRequest=new URLRequest("ch15_siteFullFlash_4b_AS2.swf");
chargeur.load(chemin);
addChild(chargeur);

// connexion avec SWF AS2
var emetteur:LocalConnection = new LocalConnection();
connexion_btn.addEventListener(MouseEvent.CLICK,AccesAS2);
function AccesAS2(evt:MouseEvent ) {
    // nom de la connexion définie dans AS2, nom de la fonction définie dans AS2
    emetteur.send("maConnexionAS2", "monActionProgrammeeAS2");
}
```

Dans le premier document (AS2), nous commençons par ouvrir une connexion avec un objet LocalConnexion :

```
// récepteur
var recepteur:LocalConnection = new LocalConnection();
```

À la suite, nous activons cette connexion, à l'aide de la méthode connect(). En paramètre, nous spécifions le nom de cette connexion invoquée plus tard dans le fichier AS3.

```
// active la connexion
recepteur.connect("maConnexionAS2");
```

Puis, sur cet objet, nous définissons une fonction dont le nom (monActionProgrammeeAS2) sera également invoqué plus tard dans le document AS3.

```
// définition de la fonction AS2 activée depuis le document AS3
recepteur.monActionProgrammeeAS2=function () {
    // actions AS2
    symbole_mc._rotation+=20;
}
```

Pour terminer, l'instruction applique une rotation de 20° sur le clip symbole_mc placé sur la scène principale (symbole_mc._rotation+=20).

Dans le document AS3 appelant, dans la fenêtre des actions, nous commençons par charger le fichier SWF codé en AS2. Puis, plus loin, nous ouvrons également une connexion en définissant un nouvel objet LocalConnexion :

```
// connexion avec SWF AS2
var emetteur:LocalConnection = new LocalConnection();
connexion_btn.addEventListener(MouseEvent.CLICK,AccesAS2);
```

```
function AccesAS2(evt:MouseEvent) {  
    // nom de la connexion définie dans AS2, nom de la fonction définie dans AS2  
    emetteur.send("maConnexionAS2", "monActionProgrammeeAS2");  
}
```

À l'intérieur de la fonction appelée par le gestionnaire d'événement en AS3, nous associons, à l'objet émetteur, la méthode `send()`. Cette méthode envoie une requête qui invoque le nom de la connexion (`maConnexionAS2`) et la fonction (`monActionProgrammeeAS2`), préalablement définies dans le fichier AS2.

En publiant le document, et en cliquant sur le bouton du document AS3, la liaison entre les deux fichiers est établie.



Il est possible d'appeler un document hébergé sur un autre serveur et d'un autre domaine que celui où figure le fichier appelant. Vous devez, pour le document appelé, ajouter l'instruction qui autorise le lecteur à communiquer avec un autre domaine. Pour cela, entre la définition de l'objet `LocalConnexion` et l'activation de la connexion, ajoutez une des deux instructions suivantes.

Si le fichier AS3 est sur un domaine différent mais identifié, inscrivez :

```
recepteur.allowDomain("http://www.nomdusite.com");
```

Ou bien, si le document AS3 est sur un domaine différent mais inconnu, inscrivez :

```
//recepteur.allowDomain("*");
```

À retenir

- Les documents AS1/AS2 peuvent être importés dans un document AS3 à l'aide d'un simple chargeur, mais aucune interaction n'est possible avec le document importé.
- Pour autoriser l'interaction du document AS1/AS2 chargé, nous devons établir une connexion à l'aide d'un objet `LocalConnexion`. Le document AS3 peut alors invoquer des fonctions développées dans le fichier AS1/AS2, d'une ancienne version de langage.

Synthèse

Dans ce chapitre, vous avez appris à redistribuer les contenus à l'écran, en fonction des dimensions d'une fenêtre, et à projeter ces contenus dans un affichage de type Plein écran, tout en permettant à l'utilisateur de restaurer l'affichage initial pour la réalisation d'un site full Flash. Vous avez également appris à gérer l'affichage de la typographie, à l'intégrer si nécessaire et à la partager, ainsi que des symboles, avec d'autres documents. Vous savez aussi comment importer physiquement un document PDF pour la publication et éviter de sortir d'un univers tout en Flash. Vous avez enfin appris à interagir depuis un document AS3, avec un contenu développé en ActionScript 1 ou 2. Vous êtes à présent en mesure de réaliser des systèmes d'affichage sophistiqués et optimisés, qui valorisent les contenus en Flash, au-delà de la surface d'affichage astreinte aux dimensions utiles d'une fenêtre de navigateur, toutes versions de Flash confondues.

16

Les solutions de référencement du Flash

Introduction

Flash a pendant longtemps été réputé pour offrir un contenu totalement opaque pour les moteurs de recherche. Mais, voici la chose résolue. Google et Adobe collaborent maintenant pour favoriser la reconnaissance des contenus édités au format Flash.

Depuis le printemps 2008, tout contenu Flash structuré selon les recommandations de Google est officiellement enregistré et présenté de la même manière que tout autre contenu HTML, aussi stricte en soit la mise en forme. Nous détaillons, dans ce chapitre, les recommandations émises par Google pour favoriser l'indexation d'un contenu réalisé en Flash.

Nous abordons aussi d'autres aspects comme : le *pageRanking* qui révèle la pertinence d'un site en fonction de sa popularité autant que par son contenu ; les métadonnées XMP ajoutée de manière intrinsèque aux contenus Flash ; l'intégration XHTML stricte plus favorable à la lisibilité d'un contenu Flash ; et l'affichage de contenu HTML alternatif pour les utilisateurs qui ne disposent pas du lecteur Flash.

À l'issue de cette annexe, vous saurez rendre vos réalisations parfaitement visibles pour les moteurs de recherche. Vous saurez même comment vous positionner mieux que des contenus réalisés en HTML simple.

Intégration XHTML stricte

Les annuaires sont sensibles à l'accessibilité des contenus. La manière d'intégrer les fichiers Flash favorise donc le positionnement du site.

Lors de l'intégration d'un fichier SWF dans une page HTML, Flash et Dreamweaver utilisent les balises `<embed>` et `<object>`. La balise `<embed>` ne fait pas partie des spécifications des normes strictes d'intégration XHTML. Pour intégrer le Flash proprement, il est préférable d'employer uniquement la balise `<object>`.

Le code de base employé pour une intégration stricte du Flash est :

```
<object type="application/x-shockwave-flash" data="monDocumentFlash.swf"
      width="800" height="600">
  <param name="movie" value="monDocumentFlash.swf" />
  <p>Texte secondaire alternatif</p>
</object>
```

Il est à préciser que la balise `<object>`, employée seule, active aussi un blocage dans Internet Explorer qui demande à l'utilisateur de confirmer l'activation du contenu en Flash avant de poursuivre sa lecture (comme contrôle Active-X). D'autres solutions d'intégration plus

souples sont disponibles pour Flash. Nous les abordons en fin d'annexe, avec l'utilisation du kit *SWFObject2*.

À retenir

- L'intégration stricte de Flash avec la balise `<object>` favorise indirectement son indexation auprès des moteurs de recherche.
- L'utilisation de la balise `<object>` implique un blocage dans Internet Explorer.
- D'autres solutions d'intégration sont disponibles avec *SWFObject2*.

Structurer un document pour l'API de Google

Depuis juin 2008, Google référence officiellement le Flash (lire <http://googlewebmaster-central.blogspot.com/2008/06/improved-flash-indexing.html>). Les résultats des pages de recherche démontrent que le contenu véhiculé dans un document Flash est désormais bien pris en compte (voir Figure 16.1).

Figure 16.1

Comparaison
des résultats
de recherche
Google, avant et
après 2008.

Indexation d'une animation SWF avant la nouvelle API de Google

Deep Impact
www.jpl.nasa.gov/multimedia/deep-impact/index-flash.html - 3k - [Cached](#) - [Similar pages](#)

Indexation d'une animation SWF depuis la nouvelle API de Google

Deep Impact
NASA's Hubble, Spitzer and Chandra Space Telescopes will be recording these ... ORBIT
PATHS This animation shows the trajectory of **Deep Impact** and the orbit ...
www.jpl.nasa.gov/multimedia/deep-impact/index-flash.html - 3k - [Cached](#) - [Similar pages](#)

Pour que les documents Flash puissent être correctement indexés, ils doivent obéir aux spécifications suivantes :

- Tous les textes contenus dans un document SWF sont enregistrés, dès lors qu'ils sont visibles à l'écran et édités dans un champ de texte statique, dynamique ou de saisie. Les textes placés dans des symboles, dans des boutons sont également pris en compte. Seuls les textes images ou vectorisés ne sont pas identifiés.
- Les URL affichées dans le texte, si elles sont cliquables, sont également enregistrées. Elles peuvent être actives à partir des fonctionnalités de lien définies depuis l'Inspecteur de propriétés, ou en ActionScript, toutes versions du langage confondues.
- Les fichiers SWF, XML, HTML et textes externes, liés à un contenu Flash identifiable (un texte) sont également enregistrés. Important, Google fait même ressortir la page HTML qui contient le Flash. Donc, non seulement Flash n'est pas opaque pour les moteurs, mais l'utilisation de la technologie Flash favorise l'indexation d'une page HTML.
- Les images, les vidéos et les graphismes vectoriels ne sont pas, en revanche, enregistrés.

- Si les contenus image et formes graphiques ne sont pas identifiables par Google, utilisez ces formats pour véhiculer des informations au demeurant inutiles, comme le texte "chargement en cours", par exemple.
- Les documents Flash intégrés *via* une instruction JavaScript ne sont pas référencés. JavaScript apparaît comme un mur anti-référencement. Préférez une intégration stricte avec la balise <object> (voir section précédente). Mais cela provoque un blocage d'Internet Explorer. Optez plutôt pour une indexation qui se base sur un contenu alternatif (voir fin d'annexe).
- Flash distingue les pages HTML, les textes TXT et les fichiers XML appelés par le SWF du fichier SWF lui-même. Ces contenus risquent donc d'être présentés en accès parallèle aux animations Flash. Veillez à désigner clairement leur libellé pour éviter les confusions et orienter autant que possible les utilisateurs.
- Les textes bidirectionnels ne sont pas encore indexés, mais Google y travaille (Hébreu, Arabe).
- Ne cachez pas d'informations sans rapport avec les noms et les descriptions affectés aux objets (technique du spamdexing ou spam). Le fonctionnement de Google favorise toujours l'indexation des sites qui servent l'utilisateur. Tout comportement mal veillant nuisible pour l'utilisateur est donc également nuisible pour votre positionnement.
- Une organisation sémantique de la structure du site (noms des fichiers, noms des dossiers) est primordiale. Aussi, attribuez des noms de fichiers et de dossiers en rapport avec le thème abordé, naturellement, sans caractères spéciaux, ni espaces. Les tirets sont considérés par Google comme des séparateurs sémantiques, mais ils ne sont pas toujours bien supportés par l'API de Flash dans le cas de documents SWF imbriqués.
- Vous pouvez également ajouter des métadonnées aux fichiers SWF afin d'orienter le robot. Pour en savoir plus sur cette technique, reportez-vous à la section suivante.
- Utilisez une mise en page XHTML claire pour le robot, qui lui permettra de distinguer sémantiquement le positionnement du contenu en Flash des autres contenus. Pour cela, utilisez des balises div dont les identifiants se rapportent aux données véhiculées (Par exemple, utilisez un conteneur HTML <div id="flash">...</div> pour contenir le Flash et <embed id="animation" ... /> pour afficher l'animation Flash).
- Pensez à toujours associer un contenu alternatif complémentaire au format HTML. Cela peut être une note d'intention du site, un résumé, une version mobile, un document PDF. Tout texte qui accompagne le Flash aide à positionner le ou les fichiers présentés.
- Vous pouvez également élaguer les éléments inutiles qui ne doivent pas être pris en compte lors de l'indexation. Filtrez pour cela les contenus utiles en utilisant un fichier "robot.txt" (consultez l'adresse <http://www.google.com/support/webmasters/bin/answer.py?hl=fr&answer=156449> pour en savoir plus sur l'utilisation du fichier robot.txt).
- N'hésitez pas à consulter : <http://www.google.com/support/webmasters/bin/answer.py?hl=fr&answer=72746> pour en savoir plus.

Pour évaluer au mieux la manière dont votre projet peut être interprété par Google, pensez que Google favorise toujours les contenus au service de l'utilisateur. Si vous apportez des contenus utiles, bien organisés, faciles d'accès, ils seront mis en avant, même en Flash. Si vous tentez de manipuler l'utilisateur, d'agir à son insu, de forcer certains contenus, de cacher des textes en les mettant blanc sur fond blanc, cela pénalisera votre positionnement. Google se met toujours à la place de l'utilisateur pour vérifier un contenu en agissant dans l'intérêt de l'utilisateur. En respectant cette logique, vous améliorerez sensiblement le positionnement de vos réalisations.



Le pageRanking. Le pageRanking désigne la popularité d'un site et principalement, la qualité des liens extérieurs qui y font référence. Cette donnée représente 50% des critères qui aident Google à déterminer le positionnement d'un site dans ses pages de résultat. Il en résulte que, même un site entièrement opaque et pour lequel aucun effort d'optimisation n'est déployé, parvient très bien à apparaître en tête du placement pour une requête ciblée. Un site 100 % HTML strict ne garantit donc pas, à lui seul, un positionnement meilleur qu'un site tout en Flash, même opaque. Néanmoins, nous ne saurions que vous conseiller de favoriser l'accessibilité de votre projet et son optimisation, afin de répondre au mieux aux différents besoins des utilisateurs.

Une excellente illustration de l'effet du pageRanking est cette anecdote apparue pendant les élections présidentielles de 2007, en France. Une équipe de webmasters avait placé au sein de nombreux blogs et forums de discussion, des hyperliens qui pointaient vers une biographie du personnage Iznogoud, personnage, ô combien réputé pour son goût prononcé du pouvoir. Ironie de l'histoire, en associant au libellé de ce lien, l'expression "Nicolas Sarkozy Président", ces webmasters ont prouvé la toute puissance de Google. Quelques semaines plus tard, il suffisait en effet de saisir "Nicolas Sarkozy Président" dans le moteur de Google, pour que la première page proposée soit la fiche biographique du personnage Iznogoud. Le mécanisme ayant opéré, ils n'avaient pas tardé à proposer l'inverse : un lien de libellé Iznogoud qui pointait vers la page de candidature officielle du candidat Nicolas Sarkozy. Sans évidemment porter de jugement sur le fond de cette forme de blague contemporaine, le résultat obtenu a prouvé combien le mécanisme de Google prenait davantage en considération, le libellé et l'emplacement des liens sur l'Internet, que les contenus qui y sont associés, eux-mêmes. C'est la raison pour laquelle nous pouvons affirmer aujourd'hui qu'un site réalisé intégralement en Flash, pour lequel une campagne virale bien étudiée a été déployée, peut obtenir un positionnement bien supérieur au référencement naturel obtenu par un site XHTML strict (voir Figure 16.1).



Le débat Flash vs. XHTML. Flash ne se positionne pas comme une technologie concurrente à la création de sites XHTML stricts. Ces techniques sont parfaitement complémentaires, au contraire. L'une apporte l'accessibilité (HTML), l'autre la richesse et les fonctionnalités (Flash). À titre d'exemple, une entreprise pourra réaliser un site 100 % XHTML pour offrir un service générique et institutionnel pérenne qui véhiculera son image "corporate", une forme de "service public" en somme. Elle peut l'accompagner, plus ponctuellement et régulièrement, sur des périodes plus courtes, avec des mini sites événementiels tout en Flash. Cela lui permet, tout en apportant un service de base, élémentaire, de valoriser et mieux communiquer aussi sur ses produits en proposant des fonctionnalités rich-médias irréalisables en XHTML. Les deux conceptions se complètent donc parfaitement. Il n'y a pas de camp tout désigné qui puisse se vanter d'offrir une solution meilleure qu'une autre. Chaque technologie apporte une valeur ajoutée utile et complémentaire l'une de l'autre.

À retenir

- Google sait enregistrer les textes contenus dans un document SWF.
- Les fichiers qui accompagnent un document Flash, importés dans le SWF ou environnants (page HTML qui le contient) sont valorisés.
- L'intégration du Flash avec JavaScript favorise le blocage avec Internet Explorer. Cette solution permet, cela dit, de positionner le Flash à l'aide d'un contenu alternatif.

Les métadonnées des fichiers SWF

Les métadonnées des pages HTML sont des facteurs importants pour le référencement, même si ces derniers sont de moins en moins pris en considération, à la faveur des contenus eux-mêmes. Lorsque les métadonnées accompagnent les contenus, en intégrant des textes et des mots qui soulignent le sens réel des informations affichées à l'écran, elles valorisent le positionnement des pages.

L'ajout de métadonnées est désormais possible pour les documents SWF, comme nous le faisons pour les documents HTML. Depuis Flash CS4, il suffit de renseigner un tableau de descriptions codé au format XMP, puis, de publier le document, pour que les informations soient automatiquement intégrées à l'animation. Dès que le robot identifie le fichier SWF, il analyse ces métadonnées et affine le positionnement du document.

1. Pour ajouter des métadonnées à un document Flash, dans Flash, faites Fichier > Informations.
2. La table des métadonnées s'affiche (voir Figure 16.2).

The screenshot shows the 'Informations' (Metadata) window in Adobe Flash CS4 Professional. The window title is 'ch16_referencementFlash_1 fla'. The window has a tabbed interface with the following tabs: Description, IPTC, Données de la caméra, Données vidéo, Données audio, SWF mobile, Catégories, Origine, DICOM, Historique, Illustrateur, Avancé, and Données brutes. The 'Description' tab is selected. The fields are as follows:

- Titre du document : Papillon
- Auteur : Arzhur CAOUISSIN
- Fonction de l'auteur : Auteur
- Description : Animation d'un papillon en couleur
- Note : ★ ★ ★ ★ ★
- Auteur de la description : Arzhur CAOUISSIN
- Mots-clés : Flash, papillon, animation
- Etat du copyright : Inconnu
- Notice de copyright :
- URL d'informations sur le copyright :
- Date de création : 09/09/2009 - 19:38
- Date de modification : 28/11/2009 - 14:03
- Application : Adobe Flash CS4 Professional
- Format : application/vnd.adobe fla

At the bottom, there is a 'Powered by xmp' logo and buttons for 'Importer...', 'Annuler', and 'OK'.

Figure 16.2

Aperçu de la fenêtre Informations.

3. Dans cette fenêtre, différents onglets sont disponibles :

- L'onglet Description permet de renseigner le profil général de l'auteur et d'informer les robots sur les caractéristiques générales du document.
- L'onglet IPTC organise plus en détail les informations relatives à la signature du document, selon la norme conventionnelle internationale employée dans le secteur de la presse.
- Les autres onglets permettent d'ajouter des informations en rapport avec d'autres systèmes d'indexation et d'autres formats de fichiers.
- À l'exception de l'onglet SWF mobile, réservé pour la publication des documents Flash à destination des appareils mobiles, nous n'en aurons pas usage pour la création de sites web.

4. Renseignez les champs requis dans les différents onglets. Puis validez.

5. Publier le SWF en faisant Ctrl+Entrée (Windows) ou Cmd+Entrée (Mac).

Le fichier SWF contient ses propres métadonnées. Si vous disposez de la Creative Suite Adobe, vous pouvez lire ces données depuis Bridge. Naviguez alors jusqu'au fichier SWF pour lire les métadonnées (voir Figure 16.3).

Figure 16.3

Affichage des métadonnées dans Bridge.



Il est possible de générer des métadonnées *via* ActionScript à l'aide de la méthode `setMetadata()`. Une documentation en ligne présente les commandes disponibles à l'adresse suivante : http://help.adobe.com/fr_FR/Flash/10.0/ExtendingFlash/WS5b3ccc516d4fbf351e63e3d118a9024f3f-7dcf.html.

À retenir

- Pour ajouter des métadonnées dans un fichier SWF, nous utilisons la fenêtre d'information du document Flash.
- Ces métadonnées sont lisibles par Google, mais aussi – et entre autres – depuis Bridge.

Gérer le contenu alternatif pour les robots et les appareils nomades

Les éléments XHTML (<div></div> ou <embed />) ne sont pas pris en compte par les moteurs lorsqu'ils sont affichés à partir d'une commande JavaScript. Pour rendre un contenu visible, il est donc préférable d'organiser son affichage directement à partir de ces balises XHTML sans les envelopper de comportements JavaScript (voir aussi http://fr.wikipedia.org/wiki/Web_profond). Mais, bien que le contenu affiché par une instruction JavaScript neutralise l'accès au document Flash, il offre un certain avantage.

À défaut de vouloir rendre le document Flash indexable pour l'API de Google, vous pouvez aussi afficher un contenu texte XHTML, alternatif au Flash. Dans ce contexte, c'est justement le contenu HTML alternatif qui va aiguiller le robot. Cette technique offre aussi un deuxième avantage. Nous pouvons afficher un contenu alternatif au Flash à destination des appareils nomades non équipés du lecteur. Cela permet, par exemple, de proposer un service différencié et davantage ciblé pour les utilisateurs d'appareils nomades. Certains appareils continuent, en effet, de ne pas pouvoir lire le Flash. Cette section apporte les solutions alternatives pour ces utilisateurs.



Les contenus alternatifs intelligents. Par le passé, nous développons un clone du site Flash, en guise de contenu alternatif. Une aberration qui a été repensée avec l'arrivée des appareils mobiles et la vague du Web 2.0. Lorsque vous utilisez un support différent d'un ordinateur classique (téléphone mobile), vous attendez un contenu approprié et qui réponde au mieux à votre besoin au moment où vous consultez. Il paraît donc plus cohérent de cibler effectivement l'affichage des contenus en fonction de la plateforme qui l'exécute et non plus de le doubler de manière parfaitement stérile. Si le site doit être consulté sur un ordinateur, il peut offrir une navigation riche, de nombreuses fonctionnalités parfois gourmandes en ressources, sans trop de difficulté. Mais si le site doit pouvoir être visité rapidement, entre deux rendez-vous, avec une connexion payante à la minute, sur un écran de taille réduite et sur un appareil aux capacités limitées, en plein soleil, vous ne pouvez évidemment pas imposer le même service que sur le site de base. Pensez à utiliser la technique JavaScript qui distingue le contenu Flash d'un contenu HTML pour permettre également d'offrir un contenu mobile approprié, différent du site référent développé en Flash.

Installer le kit SWFObject2

Pour réaliser une page HTML conforme, qui ne provoque pas le blocage d'Internet Explorer, et qui permette de spécifier à partir de quelle version du lecteur Flash, nous préférons afficher un contenu alternatif, tout en offrant un contenu identifiable pour Google, nous utilisons le kit de développement *SWFObject2*.

Pour utiliser ce kit facilement, nous téléchargeons le script SWFObject et son installateur. Le script est à placer dans le répertoire du site. L'installateur est une page HTML/JavaScript qui génère, à la volée, le code HTML requis pour une intégration conforme de l'animation Flash. Ce kit est disponible gratuitement à l'adresse suivante : <http://code.google.com/p/swfobject/>. Téléchargez les fichiers "SWFObject2_2.zip" (le script) et le fichier "SWFGenerator_1_2_html.zip" (le générateur) (voir Figure 16.4).

Figure 16.4

Téléchargement
du kit
SWFObject2.



Dans le répertoire de votre site (en l'occurrence, dans notre dossier Exemples), décompressez directement les deux fichiers. Le dossier "SWFObject.js" peut être placé à la racine du projet. La page HTML, index.html, une fois utilisée, peut être supprimée du projet. Dans les fichiers d'exemple de cet ouvrage, nous avons renommé cette page "SWFObjectIndex.html", pour éviter toute confusion avec d'autres fichiers.

En parcourant le dossier "SWFObject/" apparu à la décompression, nous distinguons, dans ce répertoire, le fichier JavaScript "swfobject.js", ainsi qu'un installateur de mise à jour du lecteur Flash nommé "expressInstall.swf". Dans ce même dossier, deux pages HTML proposent déjà le code pour l'intégration d'un document Flash selon deux méthodes différentes. La page "index.html" affiche une intégration stricte et conforme avec des balises HTML. La page "index_dynamic.html" intègre le Flash à partir du JavaScript. Cette seconde proposition adopte un codage tout à fait conforme, mais elle offre l'avantage de neutraliser le blocage d'Internet Explorer sur les applications multimédias (avec le message "cliquez pour activer ce contrôle"). Vous pouvez donc utiliser l'un ou l'autre de ces deux fichiers pour mettre en forme votre page HTML, selon la technique de votre choix.

Vous pouvez aussi lancer dans le navigateur la page que nous avons renommée "SWFObjectIndex.html", pour obtenir un code personnalisé. Pour utiliser cet installateur, procédez comme suit :

1. Lancez la page SWFObjectIndex.html dans un navigateur.
2. Cette page affiche un formulaire dans lequel nous avons simplement à désigner le nom et l'emplacement relatif du document Flash (à partir de la page HTML qui doit contenir le Flash), ainsi que ses dimensions en pourcentage ou en pixels (voir Figure 16.5).

Figure 16.5

Aperçu du générateur dans le navigateur.

SWFObject 2 HTML and JavaScript generator v1.2

Internet TV's indicates required field

SWFObject configuration [+]

SWFObject type:

Publishing method: what is this?

Detect Flash version:

Adobe Express install: ☒ expressinstall.swf

SWF definition [+]

Flash (swf):

Dimensions: x (pixels)

Flash content of:

Done

HTML definition [+]

HTML Template:

Character encoding:

Alternate content:

Use `<script>http://www.adobe.com/go/getflplayer</script>`
 using `<script>http://www.adobe.com/go/getflplayer?download=download_swftobject2_flash_player</script>` (see Adobe Flash player)

Generated output [+]

```
<script>http://www.adobe.com/go/getflplayer</script>
<script>http://www.adobe.com/go/getflplayer?download=download_swftobject2_flash_player</script>
```

3. Le générateur affiche les noms des fichiers scripts (`swfobject.js`) et de l'installateur express (`expressInstall.swf`), comme figurant à la racine du projet.
4. Pensez d'abord à ajouter le nom du répertoire `swfobject/` qui organise ces éléments, afin que le navigateur qui exécutera la page puisse atteindre ces fichiers.
5. Choisissez un type d'intégration stricte ou dynamique.
6. Cliquez ensuite sur **Generate** afin d'obtenir le code HTML de base de votre document.
7. Puis, sélectionnez ce code et copiez-le dans une nouvelle page HTML vierge que vous nommerez `index.html`, si vous en faites votre page d'accueil (voir Figure 16.6).

Figure 16.6

Paramétrage du document HTML.

[illegible]

Dans le dossier des exemples du livre, vous trouverez la page enregistrée, avec l'option d'intégration dynamique, sous le nom de "ch16_referencementFlash_2.html".

En l'ouvrant dans le navigateur, si le lecteur Flash 10 ou supérieur est intégré, la page apparaît normalement. À défaut, la version alternative est affichée. Cette version alternative est basique. Le contenu Flash s'affiche dans un coin de la page. Pour améliorer l'intégration des contenus, nous pouvons personnaliser le document.



Tester avec plusieurs versions du lecteur Flash avec Flash Switcher. Vous pouvez tester le comportement de la page à partir de différentes versions de lecteur Flash, depuis la même fenêtre de navigateur, sans nécessairement disposer de plusieurs configurations pour ce faire. L'extension de navigateur *Flash Switcher* offre cette possibilité. Pour en savoir plus l'assistant Flash Switcher, consultez le lien suivant : http://www.sephiroth.it/firefox/flash_switcher/index.php.

Personnaliser le document

Dans le mécanisme du JavaScript SWFObject, le contenu alternatif est toujours affiché par défaut dans la fenêtre du navigateur. C'est uniquement lorsque JavaScript détecte le lecteur Flash qu'il affiche l'animation SWF (voir Figure 16.7). Le cas échéant, c'est la version HTML qui est affichée. Mais nous devons bien comprendre que ce HTML est déjà en place, il n'est simplement pas visible si le lecteur Flash est actif (voir Figure 16.8). Pour cette raison, ce mécanisme est favorable à l'indexation du contenu et peut être interprété par tous les navigateurs.

Figure 16.7

Aperçu de l'animation Flash.



Figure 16.8

Aperçu de la
page HTML
alternative.



Exemples > ch16_referencementFlash_2b.html

Exemples > ch16_referencementFlash_2 fla

Dans le document "ch16_referencementFlash_2b.html", nous avons substitué le contenu alternatif affiché par défaut par le générateur SWFObject (voir section précédente), par un texte formaté avec des styles CSS, une image, un hyperlien et des titres.

Dans le corps de la page, en lieu et place du contenu de la balise `<div id="content"></div>`, nous avons introduit le texte alternatif. Les formatages CSS, eux, sont ajoutés dans l'en-tête du document :

```
<style type="text/css">
<!--
html {
    height: 100%;
}
body {
    padding:0px;
    margin:0px;
    font-size: 16px;
    font-family:Arial, Helvetica;
    background-color: #071B22;
    color: #FFF;
    background-image: url();
    background-repeat: repeat-x;
    height: 100%;
    overflow: hidden;
}
#donnees{
    height: 90%;
    width:550px;
    margin: 20px;
```

```

    position: relative;
    text-align: justify;
}
h1{
    color:#FFF;
    font-size:48px;
}
h2{
    font-size:36px;
    border-bottom-width: 1px;
    border-bottom-style: solid;
    border-bottom-color: #FFF;
}
h3{
    display:block;
    font-size: 12px;
    background-color: #333;
    padding: 3px;
    text-transform: uppercase;
}
a:link {
    color: #CCC;
}
#design a:link {
    color: #666;
}
a:visited {
    color: #CCC;
}
a:hover {
    color: #FFF;
}
#design a:visited {
    color: #666;
}
#design a:hover {
    color: #000;
}
ul, li{
    margin:0;
    padding:0;
}
li{
    display:inline;
}
#design {
    background-color: #FFF;
    color: #000;
    padding-top: 5px;
    padding-right: 10px;
    padding-bottom: 5px;
    padding-left: 10px;
    font-size: 14px;
}
#design h2 {
    font-size:18px;
    border-bottom-width: 1px;
    border-bottom-style: solid;
    border-bottom-color: #000;
}

```

Afin que le contenu Flash puisse apparaître en occupant toute la surface de la fenêtre du navigateur, nous lui avons attribué des dimensions de 100 % en largeur et en hauteur. Mais, comme nous employons un codage XHTML strict, nous devons aussi rappeler au navigateur que le corps du document et la structure HTML générale sont étendus, eux aussi, sur toute la surface de la fenêtre de navigateur. À défaut, le contenu Flash n'apparaîtra pas dans Firefox. C'est la raison pour laquelle, dans la feuille de style, nous ajoutons les attributs `CSS height:100%` pour chacun des éléments qui contiennent le Flash.

De même, afin de supprimer la barre d'ascenseur verticale qui résulte de l'intégration du Flash, nous la masquons en ajoutant, dans le style pour le corps de la page (body), l'attribut `overflow`, passé sur la valeur "masqué" (hidden). Nous pourrions aussi afficher un Flash à seulement 99 % de la hauteur, au risque toutefois d'ajouter une petite marge en bas de la fenêtre de navigateur. Vous appliquerez l'option à votre convenance. Notez cependant que l'option `overflow` induit une surface limite pour afficher le contenu alternatif. Dans ce cas, si un contenu important doit apparaître, nous vous conseillons de créer d'abord un sommaire, puis de répartir le contenu sur différentes pages HTML successives qui elles, n'utilisent pas cet attribut.

Le contenu SWF peut désormais être affiché à 100 %. Mais pour cela, nous devons aussi ajouter des contrôles ActionScript, comme vu Chapitre 15, afin de redéfinir le positionnement et l'échelle des objets selon les dimensions de la fenêtre. Sans cela, le Flash resterait calé en haut et à gauche, sans être redimensionné. Seule la couleur d'arrière-plan du document SWF occuperait toute la surface de l'écran.

Le document Flash "ch16_referncement_2.fla", dont l'exportation au format SWF est intégrée dans notre page HTML, affiche le code suivant :

```
//----- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;

//----- actions
interface_mc.lancer_btn.addEventListener(MouseEvent.CLICK,volPapillon);
function volPapillon (evt:MouseEvent) {
    TweenMax.to(interface_mc.Papillon_mc, 3, {rotation:-360, delay:0.2,
        ease:Strong.easeInOut});
}

// mise à l'échelle pour remplissage à 100 % de la page HTML
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;

stage.addEventListener(Event.RESIZE, redimensionneSWF);
function redimensionneSWF(event:Event) {
    fond_mc.width=stage.stageWidth;
    fond_mc.height=stage.stageHeight;
    //
    interface_mc.x=stage.stageWidth/2;
    interface_mc.y=(stage.stageHeight/2)-20;
}
```

Ce code réorganise l’affichage et les dimensions des éléments dans la scène, selon la taille de la fenêtre du navigateur.

Pour en savoir plus sur la gestion élastique et flottante du contenu Flash, reportez-vous au Chapitre 15 qui traite de la création de sites Full Flash.



Les pages satellites. Les pages satellites sont des pages HTML qui reprennent des contenus textes édités initialement dans un site en Flash. Cette technique, employée jusqu’à ce que Google sache interpréter le Flash (jusqu’en 2008 donc), aidait l’indexation des sites tout en Flash. Elle n’est plus recommandée aujourd’hui, parce que Google sait maintenant lire le contenu en Flash, mais aussi parce que, comme toute technique de diversion, elle finit malheureusement par être trop souvent employée contre l’utilisateur. Ces pages ne sont donc plus vraiment prises en compte sauf lorsqu’elles servent réellement le contenu et se présentent comme des pages d’accessibilité alternatives, justement. C’est le cas pour des textes destinés aux appareils mobiles ou qui apportent des informations utiles accessibles rapidement au format texte, par exemple (adresses, coordonnées, contenu éditorial).



Pour en savoir plus sur SWFObject2. Une page française décrit admirablement les options disponibles pour la gestion d’un contenu Flash avec SWFObject2. Consultez ce lien clair et instructif pour en savoir plus sur ce sujet : <http://egypte.olympic-network.com/swfobject-francais.html>.

À retenir

- Il est possible d’afficher un contenu alternatif à Flash pour les utilisateurs qui ne disposent pas du lecteur. Pour cela, nous utilisons le script SWFObject2.
- Il existe deux méthodes conformes pour l’intégration du Flash : une méthode HTML stricte et une méthode en JavaScript pour contourner le blocage d’Internet Explorer.
- La gestion de l’affichage du Flash à 100 % de la surface de la fenêtre de navigateur se détermine en attribuant des hauteurs de 100 % aux conteneurs HTML, et en définissant des comportements d’échelle et de positionnement, en ActionScript, dans le Flash.

Synthèse

Dans cette annexe, vous avez appris à rendre indexable un site entièrement réalisé en Flash, pour favoriser son positionnement dans les moteurs de recherche. Vous avez appréhendé le mécanisme de Google dans sa recherche de contenus. Vous avez appris à gérer l’affichage d’un site pour les systèmes qui ne disposent pas du lecteur Flash et notamment certains appareils mobiles. Vous êtes en mesure de réaliser et concevoir des interfaces Flash entièrement indexables par les moteurs de recherche.

17

L'accessibilité dans Flash

Introduction

Dans cette annexe, nous présentons les outils d'aide à la conception de sites accessibles, dans Flash, pour les personnes à "mobilité réduite sur le Web", qui utilisent des logiciels d'accompagnement pour lire les contenus (lecteurs d'écran).

Si Flash est naturellement permissif aux lecteurs d'écran, à condition toutefois qu'il soit intégré de manière stricte, des options de formatage et des méthodes de conception permettent d'organiser le contenu de manière encore plus lisible.

L'organisation d'un site accessible apparaît bien souvent comme une contrainte pesante en production, mais elle permet souvent de résoudre aussi des problèmes évidents d'ergonomie. Si un site est accessible pour un lecteur d'écran, il le sera pour tout utilisateur. De plus, si un site est accessible et pertinent pour tout utilisateur, il sera aussi valorisé par les moteurs de recherche. Réaliser un site accessible offre donc de nombreux avantages qui apportent à tous les utilisateurs. Rien n'empêche cela dit, ponctuellement, de réaliser des contenus qui sortent de ce cadre drastique, si la base, elle, demeure au moins suffisamment ouverte à tous les utilisateurs.

Flash propose, dans son interface, différents outils d'aide à l'accessibilité. Les métadonnées, que nous avons développées précédemment, y concourent. Mais aussi, les titres, les descriptions, l'ordre des tabulations et des champs, ainsi que quelques classes `ActionsScript` dont nous vous présentons ici les ressources pour pouvoir les utiliser et étendre plus spécifiquement vos développements vers ce type de configuration.

Les options d'accessibilité sont utiles pour toutes les personnes "à mobilité réduite sur le Web" au sens large, qu'il s'agisse de malentendants, non-voyants, paraplégiques, mais aussi les "accidentés ponctuels de la vie domestique" voire, les utilisateurs sans handicap mais dont l'équipement informatique ne fonctionne tout simplement pas correctement. Un étranger sur un site exclusivement en français est en situation de handicap. Une personne sans plugin adéquat pour visualiser un site Flash est en situation de handicap (voir Annexe précédente). Un texte écrit trop petit face à une personne malvoyante (et qui n'utilise pas de lecteur d'écran) et dont il n'est pas possible d'augmenter la taille du texte... est en situation de handicap. Une personne habituée à naviguer en utilisant que du clavier face à une interface utilisateur exclusivement basée sur l'usage de la souris est en situation de handicap. Un site faisant appel à une vidéo sans sous-titre, pour un utilisateur en entreprise qui n'a pas d'haut parleur, le place en situation de handicap. Un site qui prévoit le chargement d'un PDF pour pallier à un souci d'accessibilité et qui fournit un PDF non accessible place l'utilisateur en situation de handicap. Et autant d'autres situations cocasses qui nous mettent tous en situation de handicap. Nous sommes donc *a priori* tous concernés.

La fenêtre Accessibilité

Vous pouvez ajouter des titres, des descriptions et associer des contrôles de navigation, à tout symbole distribué dans Flash. Il suffit de renseigner les champs mis à disposition par la fenêtre d'accessibilité, pour ce faire.



Exemples > ch16_accessibiliteFlash_1 fla

Dans ce document, la scène affiche quelques symboles dont un champ de texte, un Movie-Clip et un bouton (voir Figure 17.1).

Figure 17.1

Aperçu de la scène principale.



Vous pouvez ajouter des titres et des descriptions sur la scène et sur les objets eux-mêmes avec le menu Fenêtre > Accessibilité.

Une fenêtre apparaît et affiche différents champs de saisie. Cliquez en dehors de la scène de manière à ne rien sélectionner. La fenêtre d'accessibilité affiche alors les entrées pour la scène.

Renseignez les champs Nom et Description pour informer l'utilisateur sur le type de contenu mis en scène (voir Figure 17.2).

Vous pouvez également renseigner des informations, plus ponctuellement, pour chaque objet. Il suffit de le sélectionner et de renseigner les champs qui correspondent à votre sélection active.

Cliquez successivement sur les symboles `interface_mc`, `papillon_mc`, sur le champ de texte dynamique et sur le bouton, puis, inscrivez les textes qui décrivent succinctement ce qu'ils représentent.

Figure 17.2

Aperçu des options pour la scène principale.

☒ Rendre l'animation accessible
☒ Rendre les objets enfant accessibles
☒ Etiquetage automatique

Nom : Le vol du papillon

Description : Cette animation fait tourner un papillon à 360° sur lui-même.

Figure 17.3

Informations d'accessibilité du symbole papillon_mc.

☒ Rendre l'objet accessible
☒ Rendre les objets enfant accessibles

Nom : Papillon

Description : Papillon coloré et stylisé (forme vectorielle)

Raccourci : P

Ordre des tabulations : 2

Figure 17.4

Informations d'accessibilité du champ de texte statique.

☒ Rendre l'objet accessible

Description : Titre Le vol du papillon

Ordre des tabulations : 3

Figure 17.5

Informations
d'accessibilité du
symbole bouton.

☒ Rendre l'objet accessible

Nom : bouton Lancer

Description : Active une rotation à 360° sur le papillon

Raccourci : l

Ordre des tabulations : 1

Lorsque des symboles enfants sont distribués à l'intérieur d'un objet, l'option Rendre les enfants accessibles est proposée. Par défaut, tous les enfants sont accessibles. Mais, dans certains cas, il peut être intéressant de neutraliser cette option afin d'éviter des bruits visuels inutiles et masquer des objets purement graphiques utilisés pour l'habillage. Cela contribue à simplifier l'ossature du document et facilite l'accès aux autres contenus.

Vous remarquez que, dans certains cas, les options Ordre des tabulations et raccourci sont également disponibles.

Les raccourcis permettent à l'utilisateur du lecteur d'écran d'accéder directement à l'objet, par simple activation d'une touche du clavier. Vous devez inscrire la lettre que vous souhaitez désigner comme touche à activer pour rendre cet objet accessible. Utilisez de préférence une lettre à consonance dure qui résume la tonalité du mot ou sa première lettre. Dans notre exemple, la lettre l est employée pour désigner le bouton Lancer.

Dans le champ Ordre des tabulations, spécifiez le nombre de fois pour lequel l'utilisateur doit appuyer sur la touche Tabulation, pour activer l'élément désigné. Dans notre exemple, le bouton Lancer est associé au chiffre 1. Il faudra donc appuyer une fois sur la touche Tabulation pour activer ce bouton, une fois le module Flash préalablement activé. Lorsque vous publiez la page HTML du document Flash dans le navigateur, pour activer l'animation avec les options d'accessibilité, procédez comme suit :

1. Activez d'abord l'objet Flash en cliquant dessus ou en appuyant sur la touche Tabulation une première fois.
2. Puis, une fois le Flash actif, appuyez de nouveau sur la touche Tabulation, autant de fois que la valeur désignée dans le champ Ordre des tabulations de la fenêtre d'accessibilité.
3. Puis, appuyez sur la touche Entrée pour confirmer l'activation des instructions associées à cet objet.

Afin de garantir une ergonomie la plus cohérente qui soit, utilisez un ordre des tabulations qui corresponde autant que possible à l'ordre logique et chronologique d'exécution des événements.



Accessibilité dans Flash. Pour en savoir plus sur l'accessibilité dans Flash, consultez aussi le lien suivant : http://help.adobe.com/fr_FR/Flash/10.0_UsingFlash/WSd60f23110762d6b883b18f10cb1-fe1af6-7c3ea.html#WSd60f23110762d6b883b18f10cb1fe1af6-7c3ca.

À retenir

- L'édition de textes de description pour l'accessibilité est disponible dans Flash. Pour cela, utilisez la fenêtre Accessibilité.
- Pour restreindre l'accès à des informations utiles uniquement, il est possible, à l'inverse, de neutraliser des objets. Pour cela, nous désactivons les options d'accessibilité des objectifs actifs, depuis la fenêtre d'accessibilité.

Concevoir un document Flash pour les non-voyants et malentendants

Parmi les utilisateurs de lecteurs d'écran non-voyants ou malentendants, des recommandations spécifiques sont à suivre. Nous les détaillons dans cette section :

- Évitez de créer des boucles d'animation qui provoquent un bavardage encombrant. Lorsque la tête de lecture rejoue perpétuellement des images sur lesquels des textes sont affichés, ces informations peuvent être répétées indéfiniment et à voix haute, par les lecteurs d'écran. Cela perturbe la navigation. Pensez à appliquer un stop sur chaque image du scénario qui désigne une nouvelle rubrique. Privilégiez les conceptions de site dans le scénario, où chaque rubrique est jouée l'une à la suite de l'autre, et séparée par un stop. Les interfaces dynamiques ne sont pas encore totalement digestes par tous les lecteurs d'écran.
- Évitez d'imposer un son ou une ambiance musicale trop prégnante afin de permettre aux utilisateurs d'entendre leurs propres lecteurs d'écran lire les contenus.
- Ne vectorisez pas le texte, car les objets graphiques et les images ne peuvent pas être identifiés.
- Associez la plupart des actions à des commandes du clavier, en plus des commandes de souris. Cela permet aux personnes qui n'accèdent pas à la souris de compenser en utilisant le clavier.
- Le mode transparent et opaque, sans fenêtre, défini dans les paramètres HTML du document Flash, rendent le Flash complètement inaccessible. Évitez d'utiliser ces options.
- Pensez à associer, autant que possible, des textes à chaque objet interactif (boutons, liens, zones d'interaction) pour en faciliter la reconnaissance.



Aides au développement de contenus spécifiques pour les malentendants et les malvoyants. Des outils d'aide au développement sont disponibles à l'adresse suivante, pour Windows uniquement : http://help.adobe.com/fr_FR/Flash/10.0_UsingFlash/WSd60f23110762d6b883b18f10cb1fe1af6-7c47a.html.

À retenir

- Pour rendre un site accessible aux non-voyants et malentendants, il est recommandé d'organiser les contenus dans le scénario en privilégiant les textes.

Synthèse

Vous avez appris à optimiser un site pour l'accessibilité et rendre un contenu Flash presque universel. Vous êtes en mesure, à présent, de créer des sites pour le plus large public, tout en apportant une valeur graphique ajoutée et fonctionnelle, significative.



Ressources

Quelques ressources pédagogiques pour le développement en ActionScript 3 et les technologies associées à la création d'interfaces riches sont disponibles en ligne. Vous trouverez ci-après une liste non exhaustive des dernières publications qui complèteront idéalement cet ouvrage.

Livres

D'After Effects à Flash, de Flash à After Effects, Richard Harrington et Marcus Geduld. Éd. Pearson. DVD inclus. Couleur. 38 €.

Cet ouvrage vous accompagnera idéalement pour étendre vos connaissances sur la vidéo interactive avec des créations graphiques, réalisées dans After Effects.

L'art du bluff avec Flash CS4, Chris Georgennes. Éd. Pearson. DVD inclus. Couleur. 29 €.

Pour prendre en main les outils d'animation, de manière didactique, avec des notions pratiques d'animateurs professionnels.

Apprendre à programmer en ActionScript 3, Anne Tasso. Éd. Eyrolles. Noir et blanc. 29,90 €.

Ce livre aborde l'interfaçage dynamique sans aucun élément placé sur la scène. Il peut vous accompagner dans l'automatisation de certains processus de mise en forme, surtout pour des contenus externalisés.

Pratique d'ActionScript 3, Thibault Imbert. Éd. Pearson. Noir et blanc. 56 €.

La référence de la programmation en ActionScript 3. Ce livre contient les techniques les plus avancées en terme de développement, avec de nombreux conseils relevant de l'optimisation des ressources utilisateur. Vous y abordez aussi les entrailles du lecteur Flash pour vous familiariser encore plus avec la logique intrinsèque de Flash.

Tutoriels vidéos

www.alltutorial.com

Ce nouveau petit site pratique et prometteur propose des tutoriels vidéos entièrement gratuits et de qualité, en langue française, sur les logiciels graphiques.

Tv.adobe.com/fr

Ce site propose des tutoriels vidéos sur les produits Adobe, en français. D'autres tutoriels sont disponibles sur la version américaine, directement à l'adresse tv.adobe.com.

Sites web et forums

www.thefwa.com

Le site des références en Web design offre plusieurs centaines de liens vers les sites les plus élaborés graphiquement. Une référence incontournable.

flash.mediabox.fr

Le forum des développeurs en ActionScript. De nombreuses astuces et conseils sur vos développements.

blog.greensock.com/tweenmax

Les dernières mises à jour de la classe TweenMax, disponible directement sur ce site, et son forum, en anglais.

www.sephiroth.it

Un site communautaire sur le développement en général, dont ActionScript 3 en particulier.

sketchup.google.com/intl/fr

Le site de présentation du logiciel gratuit Google Sketchup, pour la modélisation 3D. Ce site propose aussi une bibliothèque d'objets 3D ainsi qu'une riche documentation, en français.

flash.mediabox.fr

Le forum des développeurs en ActionScript. De nombreuses astuces et conseils sur vos développements.

www.papervision3d.org

Le site de la communauté des développeurs de la classe PaperVision.

www.adobe.com/fr/devnet/actionscript

Le site communautaire des développeurs ActionScript, par Adobe. De nombreux conseils, méthodologies et solutions connexes y sont présentées.

www.yazo.net

Un site pour apprendre les bases du langage ActionScript 1, 2 et 3, avec les fichiers en libre téléchargement et la palette Yazo.

Dictionnaires

Aide contextuelle

Flash intègre une aide contextuelle pour vous aider à découvrir les définitions des différentes méthodes, propriétés, et autres éléments du langage ActionScript. Depuis la fenêtre d'Actions, faites clic-droit, puis activez l'option Aide.

www.adobe.com/support/documentation/fr/flash

Cette page recense les différentes aides disponibles pour l'apprentissage d'ActionScript, au format texte.

Index

3D 289

affichage, optimiser 268
 After Effects 147
 améliorer l'affichage des images 250
 animation
 Caurina 250
 Tween 250
 TweenMax 250
 banque d'objets 3D libres 300
 champ de vision 284
 exporter 299
 focale 284
 galerie vidéo 268
 Google Sketchup exporter 300
 livre interactif réaliste 250
 mouvements
 d'objets 311
 de caméra 304
 mur d'images 260
 native 241
 optimiser un site 3D 314
 PaperVision
 aide en ligne 328
 guide de référence 328
 interactivité avec les objets 3D 328
 lumière 328
 point de fuite 283, 284
 propriété
 rotationX 241
 rotationY 241
 rotationZ 241
 x 241
 y 241
 z 241, 279
 rendu (publication) 311
 verso des objets 3D 248
 vidéo 174
 Voir aussi **Modéliser et PaperVision**

A

AAC, codec 183
Accélération 18, 53
 matérielle 441
Accessibilité 477, 481
Acrobat 450
ActionScript 134
 différence entre AS2 et AS3 9
 importer AS1/AS2 dans AS3 458
 philosophie du langage 9
 random 101
addASCuePoint 226
addChild 275, 286
addpage() 354
Adobe After Effects 147
 exporter
 directement le projet vers
 Flash 153
 FLV 149, 154
Adobe Media Encoder 177
 audio, onglet 183
 bandes noires 160
 débit 183
 exporter en FLV 155
 Filtres (onglet) 162
 formats pris en charge 156
 fréquence 183
 Multiplexeur 162
 niveau 182
 profil 181
 recadrer 158
 réglages d'exportation 158
 vidéo, onglet 179
Adobe Photoshop
 gestion des couches 365
 Voir aussi **Photoshop**

Adobe Premiere Pro, exporter en FLV 154

ADSL 151, 164

Affichage

élastique 443
 flottant 443
 plein écran 437, 440
 AllowFullScreen 439
 quitter 443

Afficher

contenu alternatif à Flash 469
 plein écran 473
 symbole 37

Aide de Flash 111

Aliasing 250, 268, 452
 Voir **Lisser les images**

align, propriétés 446

AllowFullScreen 440, 443

alpha 396

 propriété 12

AMFPHP 135

Amortissement 18, 22, 53

Anaglyphe Voir Relief

Angle 22

Animation 7

3D (Voir **3D**) 279
 boucle 23
 détecter la fin 35
 enchaîner 36, 47
 fluidité 28
 forme 86
 intervalles dans le temps 59
 optimiser 14
 panoramique 7
 plus nette 27
 rétablir les propriétés animées
 après une interpolation 73
 Tween et TweenMax 29

Animer un livre 3D 242

API 331

Apple Compressor 147

Apple Final Cut Pro, exporter en Quick Time 154

Apple iMovie, exporter en DV ou MOV 155

Apple Motion 141
exporter en Quick Time 145

Apple Time Machine 279

Appliquer des propriétés de MovieClip à un objet 64

Armature 77

Array 384

Arrêter le scénario 58

Arrondir un chiffre 231

AS3AnimationSystem 29

Ascenseur 54
masquer 475

Assistant TweenMax 42

Astérisque (*) 33

Atteindre une image du scénario 59

Audio 161, 167, 206
AAC 183
Débit 183
Fréquence 183
volume 205

B

Barre

d'ascenseur 54
de chargement 106

BevelFilter 65, 70

Bézier 42, 44

Bibliothèque
exporter pour ActionScript 383
partagée 454

Bitmap 331, 385

BitmapData 334, 359, 385

Blender 299

BlurFilter 65, 68, 340

Bones 77, 79

Boucle 23, 59, 64
accessibilité 481
arrêter 64
for 258
réaliser une texture raccord 24

Bouton

basculer, interrupteur 442
cibler son contenu 395
conflit d'interactivité 400
états 389
interfaçable 395
masquer la main au survol 414
neutraliser, désactiver 414

bufferTime 203

byteArray 451

C

Cache vidéo 203

cacheAsBitmap 335

Cadence 10
animations Flash 165
vidéo 138, 143, 165
modifier 165

Calque, nommer 8

Camera3D 311

CaptionButton 215

Caractères
manquants 435
spéciaux 10

Carrousel 251

Carte interactive 38

catch 133, 134

Caurina 29, 250

CBR, VBR Voir Vidéo

Centre des symboles 24, 247

Chaîne de caractères,
conversion 121

Champ de texte dynamique 111,
431

Chapitrage 207

Chargement 102
contenu

chemin 103

externe 103

erreurs 131

Voir aussi **Progression et Chargeur**

Chargeur 235

d'image 103

Chronomètre 59

Ciblage

contenu
imbriqués 232, 235, 237
inexistant 403
symbole bouton 395
dynamique 259
étiquette dans Flash 423
objet dynamique 258

Cinema 4D R11 299

Cinématique inverse 77

Classes 11

après la compilation 42
AS3AnimationSystem 29
Caurina 29
centraliser 42
compiler 102
contextMenu 417
Filters 57, 65
ik 77
getArmatureAt() 82
IKJoint 77
registerElements() 83
importer 11, 32, 41, 51, 290
installer 428
localiser sur le système 32
mécanisme 29
Timer 57, 59, 60
transitionManager 57, 59, 63
Twease 29
Tween 29, 34
TweenEvent 35
TweenMax 29, 38

classPath 42

Clavier

accessibilité 481
commandes inactives en mode
plein écran 441
interactivité 315

navigation 320
tableau des valeurs des touches 323

Clip Voir **MovieClip**

Codage différentiel 180

Codec vidéo 137

colorMatrix 343

ColorMatrixFilter 345

ColorTransform 360

Communication Flash/HTML 417

Compilation 29, 33

Composant
FLVPlayback 169
paramétrer 174
ScrollBar 431

Compter
nombre d'images dans le scénario 245
nombre d'objets dans un symbole 258

concat 346

Concaténation 346

condenseWhite 434

Conditions 14, 18, 393

content 235

contentLoaderInfo 107

Contenu alternatif 469

contextMenu 417, 419, 420

ContextMenuBuiltInItem 419

Contrôle du temps 59

Conversion degrés et radians 22

Convertir un fichier KMZ en DAE 303

Convertir un objet 64
en **MovieClip** 64

copyChannel 385

Couche, extraire 385

Couleurs
décimale 356
hexadécimale 356
modifier 356

nuancier, créer 357
prélever 360
puits, créer 357

CoverFlow 251

Crénelage 250, 268, 452
Voir **Lisser les images**

CSS, dans Flash 430

CUE_POINT 226

CuePoints 211, 217

currentFrame 376

currentTarget 129, 131, 392

Curseur de défilement 54

D

Débit 183

Décalage 53

Décélération 18

Défilant 48

Déformation 13

Dégrouper 451

Déplacer un symbole (startDrag) 406

Détecter
fin d'une animation 35, 47, 64
image du scénario 376
version du lecteur Flash 469

Développement
back-office 134
front-office 134

Dictionnaires 484

Différence (opérateur !=) 267

Dimensions
d'une page web 139
de la fenêtre de navigateur 437

displayState 442

Document, organiser 8

Données, gérer avec XML 121

draw 385

DropShadowFilter 65, 69

DSMax 299

E

easing 32

Échantillonner la vidéo pour Flash 153

Écouteur 103
arrêter 55
placer 63

Effets 26, 57
amortissement 22
fenêtre 149
ralenti 21
reflet 254
relief 377
spéciaux 141, 147, 153

Élasticité Voir **Affichage flottant**

Embed 463

Enchaîner
des actions 266, 412
les animations 35

ENTER_FRAME 52, 207, 228

Enveloppe vidéo 173

Ergonomie 48

Erreur
chargement 131
exécution 131

Espace mémoire
optimiser 14
typage 20

Étendre les propriétés d'un objet 64

Étiquette 423
créer 426

Event 20

Event.ENTER_FRAME 10, 14

Exécution, mode pour les squelettes 92

Extension 10, 14
.as 102
de classe 64

F**F4V** 162

Quick Time 184

Voir aussi Vidéo**filters** 65, 264, 345

GlowFilter 268

Filtres 65, 264, 345

animation et interpolation 352

appliquer 343

biseau 70

Bleu 348

correction des couleurs 343

Cyan 348

décliner 71

flou 68, 340

halo 69, 70, 268

Jaune 348

Magenta 348

Mélangeur de couleurs 348

négatif 347

ombre portée 68, 69

par lot 351

propriétés 73, 75

Rouge 347

saturation 345

Vert 347

finally 133**fl** 32**Flash**

structurer un document 464

vs. HTML 466

FlashPaper 451**Flottement, effet** 19**FLV** 162**FLVPlayback** 169, 174, 187

ActionScript 2 et 3 170

autoPlay 273

BackButton 190

boutons séparés (*voir* Vidéo)
189

BufferingBar 190

CaptionButton 190

contentPath 170

fenêtre 169

ForwardButton 190

FullScreenButton 190

MuteButton 190

PauseButton 190

PlayButton 190

PlayPauseButton 190

Preview 174

scaleMode 273

SeekBar 191

skin 172

source 170, 273

stop 273

Stopbutton 191

VolumeBar 191

FLVPlaybackCaptioning 211**Focus** 413**Fonction**

interrompre 274

neutre 60

nommer 10

paramètre 60

Fonctionnalité applicative 48**for** 127, 131, 258**for each... in** 259**Formats vidéo** 140**Forme Voir Animation****FreeCamera3D** 311**Fréquence** 183**FTP** 168**Full Flash** 437**G****Galerie**

animer 29

d'images 99

voir Image

vidéo 3D 268

getChildAt 237, 258, 286**getPixel** 359, 360**gKaster** 177**GlowFilter** 65, 70**Google** 464**Google Sketchup** 300**gotoAndStop()** 59**Graphisme** 331**Greensock** 41**Greensock TweenMax** 73**H****H-264, codec** 179

niveaux 182

hideBuiltInItem() 419**Hiérarchie, cinématique** 77**Historique de navigation dans**

Flash 428

HTML 134, 465

importer dans Flash 430

lien vers Flash 423

tester les liens 427

voir aussi Intégration

XHTML stricte

htmlText 434**Hyperlien** 421**I****i, variable d'incrément**

111, 118

ik

getArmatureByName() 82

getBoneByName() 84

getChildAt() 83

headJoint 77

IKArmature 78, 82

IKBone 77, 82

IKEvent 82

IKJoint 82, 83

IKManager 78, 82, 92

IKMover 82, 84, 94

JKMover 78

tailJoint 77

Image

aléatoire 99

carroussel 251

externalisée 108

externe 99, 103, 107

- galerie 99
 - XML 113
- mur 3D 260
- réaliser une texture raccord 24
- tremblante, éviter 27
- Image de scénario Voir Scénario**
- iMovie** 155
- import** 32
- Importer**
 - classes ActionScript 290
 - CSS dans Flash 430
 - HTML dans Flash 430
 - PHP/MySQL 134
 - police 454
 - un squelette 94
 - variable 434
- Imprimer un SWF** 353
- Incrémentation** 26, 53, 59, 111
- InDesign** 450
 - réaliser un livre au format SWF 250
- Indexation** 464
- Initialisation** 73
- int** 20, 28
- Intégration XHTML stricte** 463
- Interactivité** 315
 - objets dynamiques 122
 - résoudre les conflits d'objets imbriqués 400
- Interface élastique flottante** 443
- Internet Explorer, blocage des contrôles active-X** 469
- Interpolation** 34
 - durée 266
 - interrompre 36
 - Tween 36
 - TweenMax 45
- Interrompre**
 - fonction 274
 - interpolation 413
 - un contenu chargé 235
- Inverse kinematic Voir Cinématique inverse**
- IO_ERROR** 132
- IOErrorEvent** 132, 134
- ips Voir Cadence**
- Itération** 258
- J**
- Jauge de chargement** 103
- JavaScript** 134
- Jouer le scénario** 59
- K**
- KEY_DOWN** 320
- KEY_UP** 320
- keyCode** 320
- L**
- Label Voir Étiquette**
- Langage, nommer les calques automatiquement** 8
- Lecteur Flash**
 - menu contextuel 417
 - pénétration (statistiques) 287
- Lecteur vidéo**
 - H-264 pour Flash 6 et suivants 191
 - personnalisé 188
- Lévitiation** 18
- Lien**
 - HTML 421
 - tester 427
 - vers Flash 423
- Lipsync** 86
- Lissage** 250
- Lisser**
 - graphismes vectoriels 335
 - images 331
 - vidéo 268
- Liste d'affichage** 82
 - modifier l'ordre 275
- Livres** 483
- load** 102
- load()** 118
- Loader** 100
- localConnexion** 458
- M**
- mask** 340
- Masque**
 - conflit avec les polices de caractère 454
 - dynamique 340
 - progressif (bords flous) 340
- Masquer**
 - pointeur Main au survol d'un bouton 35
 - symbole 37
- Math** 22
 - ceil 231
 - random 356
- Math.random** 103
- Math.random()** 101
- Maya** 299
- Mémoire, optimiser** 9, 14
- Menu**
 - contextuel 417
 - déroulant 407
- MENU_ITEM_SELECT** 420
- metaDataEvent** 226
- Métadonnées** 467
 - créer dynamiquement 468
- Microsoft Aero** 279
- Microsoft Window Movie Maker, exporter en AVI-DV** 154
- Mise en cache vidéo** 139
- Mode plein écran** 437
- Modéliser** 299

MouseEvent 37
 MOUSE_OVER 37
 MOUSE_UP 55
mouseX 21
Mouvement
 caméra 3D 304
 organique 86
 progressif 18
MovieClip 64
 afficher la main au survol 414
 conflit d'interactivité 400
 convertir 64
 fonction Bouton 389
 neutraliser, désactiver 414

N

name, propriété 83
navigateToURL 421
Navigation, repères 218
NetStream 195
Nettété 27
 images tremblantes 27
 voir aussi **Lisser**
new Tween() 34
nextFrame 377
Nodes 79
Nombre d'images par seconde 10
Nommer
 fichiers et dossiers 465
 fonction 10
 occurrences 10
Nancier 357
Number 9, 16, 20, 28
numChildren 258

O

Object 463
Objet
 opacité 12
 rotation 12
 visibilité 12

Occurrences, nommer 10
On2 VP6 137, 162, 163
onCompleteListener 47
onStartListener 47
Opacité 396
 accessibilité 481
opérateur != 267
Optimisation mémoire 9
Organiser
 le code 9, 16
 les fichiers du site 102
overwrite 413

P

pageRanking 466
Pages satellites 476
Panoramique 7
 défilement en boucle 23
 réaliser une texture raccord 24
 sens de défilement 25
PaperVision 289
 différence entre
 Camera3D et
 FreeCamera3D 311
 scène 3D et zone d'affichage
 309
 exemple en ligne 314
 importer
 MovieClip dans l'espace 3D
 309
 objet 3D DAE 308
 installer 289
 intégrer 297
 nativement 297
 ponctuellement 298
 nombre de polygones 300
 pitch 314
 placer une scène 3D entre deux
 MovieClip 309
 primitives 320
 roll 314
 yaw 310, 314
 Z (propriété) 311

zone d'affichage de l'espace 3D
 309
 zoom 311
parseCSS 433
Particules 141, 149
Pas
 d'accélération 12
 d'incréméntation 26
Passes d'encodage Voir Vidéo
PDF 450
Performance d'affichage 441
perspectiveProjection
 filedOfView 284
 focalLength 284
 projectionCenter 283, 284
Photoshop 24
pitch() 314
Pixelisation 250
Pixellisation Voir Lisser les images
Placer un objet au premier-plan
 275
playheadTime 204
Plein écran Voir Affichage
Podcast 168
Poids standard d'une page web
 103
Point(X,Y) 84
Pointeur, aspect du pointeur au survol 35
Points de repère 211
 voir aussi **Vidéo** 217
Police
 conflit avec les masques 454
 importer 451, 454
 vectoriser 451
Position
 calcul 341
 d'un clip 12
 relative 412
prevFrame 377
PrintJob 353
Progression du chargement 105,
 107

Propriétés 12

- alpha 47
- animer 7, 14, 34, 44
- courantes 14
- rotation 45, 46
- scaleX, scaleY 47
- visible 37
- y 46, 52

Pseudo extension 10**Puits de couleurs 357****push() 384****Q****Qualité 250****Qualité d’affichage Voir Lisser les images****Quick Time 184****R****Racine (root) 237****Ralenti 18**

- effet 21

Rastérisation 250**Ratio des pixels 138****Rebondissement 18****Rectangle 54****Redimensionner la fenêtre 446****Référencement 463, 464****Relief 361**

- anaglyphe, créer
 - à partir de 2 images 370
 - ActionScript 375
 - Photoshop 363
- couches RVB 365
- interfaçage dynamique 377
- lunettes
 - actives 361
 - passives 361
- réseau lenticulaire 361
- technique de prise de vue 362

vidéo 377

- zone
 - cadre 363
 - fenêtre 363, 365
 - jaillissante 363, 367

removeEventListener 14, 274**Répartir vers les calques 8****RESIZE 446****Résolution 139, 437**

- limite de Flash 349

Ressources 483**Restauration 73****Retard 18, 20****Retarder une action 59****Richmédia 48, 239****roll() 314****RollOver vidéo 276****root (racine) 237****rotation, propriété 12****RVBA, extraire une couche 385****S****scaleMode 445****scaleX, propriété 12****scaleY, propriété 12****Scénario 7**

- atteindre l’image
 - précédente 377
 - suivante 377
- compter les images 245
- déteindre l’image active 376

Scénariser 48**Scène**

- hauteur 264
- largeur 264
- mode d’affichage 445
- propriété 3D 284

SCPlugin 292**Scroll 54****seek 204, 207****seekToNavCuePoints 218****Séparer 451****setMetadata 468****Sites web et forums 484****Sketchup 304****Skin 188****smoothing 250, 333****Son, accessibilité 481****Sorenson Spark 137, 162, 163****Sous-classe 64****Sous-titrage 211****Spamdexing (spam) 465****Squelette 77**

- animer 84
- avec formes graphiques 86
- charger dans un nouveau SWF 94
- construire 80
- contrainte de mouvement et de rotation 80
- définir 81
- humain 82
- mode Exécution 80
- nom d’occurrence 80
- ordre d’affichage des liaisons 81
- programmer 77

stage 237, 445**stageDisplayState 443****stageHeight 264, 445****stageWidth 264, 445****startDrag 406****startDrag, stopDrag 54****Statistiques, pénétration du lecteur Flash 287****Streaming 210****String() 434****Structure**

- conditionnelle 28
- d’un document Flash 235
- flottante 476

Styles 214, 215

- CSS 473

Suivre le pointeur 18

Supprimer

écouteur 113

symbole 451

un SWF chargé 232

Surface utile 139, 437

SVNX 292

swapChildren 247

SWFAddress 428

SWFObject 428

SWFObject2 469, 476

Swift 3D 299

Switch... case 393

Symboles

déplacement 25

exporter pour ActionScript 383

nom d'occurrence 8, 14

placer le centre 24, 247

position 27

transparent 48

zone active 48

Syntaxe chameau 14

séparateurs 10

Systèmes de navigation 389

T

Tableau 384

target 129, 131, 210, 392

Technologie 466

Teinte aléatoire 355

Temporiser une action dans le temps 59

Tête de lecture, boucle 420

Text Layout Framework 458

Texte 464

accessibilité 481

défilant 54

dynamique 431, 452

statique 452

vectoriser 451

Voir aussi **Police**

Timecode 208, 221, 226

Timer 59, 93, 286

détecter la fin 64

stopper 64

Tortoise SVN 290

totalFrames 245

Touche du clavier Voir Clavier

trace 131

transitionManager 60, 62, 64

définition 62

paramètres 62

Transparence 396

accessibilité 481

animer 13

du Flash 435

Transtypage 64

chaîne de caractères String()
434

Trucage vidéo 141, 152

try 133, 134

Tutoriels vidéos 483

Twease 29

Tween 29, 32, 38, 250

enchaînement 35

instabilité 36

interpolation 36

interrompre 36

TweenEvent 32, 35, 38, 47

COMPLETE 47, 90

enchaînements 36

MOTION_CHANGE 36

MOTION_FINISH 35, 36

MOTION_LOOP 36

MOTION_RESUME 36

MOTION_START 36

MOTION_STOP 36

TweenLight 41

TweenLite 29

TweenMax 29, 44, 230, 250

assistant 42

bézier 42

carte interactive 38

courbe de bézier 44

définition 45

delay 73

durée 266

enchaîner des interpolations à
47

importer la classe gs 67

mise à jour 41

onCompleteListener 412

onStartListener 412

overwrite 413

propriétés 73

TweenMax gs, importer 67

Typage 14, 28

nombre 20

Typographie 451

voir **Police** 454

U

UILoader 110, 112

uint 20, 28

unloadAndStop 235, 238

URL 421

URLLoader 118, 119

URLRequest 118

URLVariables 434

useHandCursor 35

V

Valeur

aléatoire 356

générer 101

incrémenter 26

Variabilité du débit Voir Vidéo

Variables 18, 28, 33, 51

globale 116

importer dans Flash 434

locale 116

si 59, 61

valeur par expression 35

VBR, CBR Voir Vidéo

Vectorisation, accessibilité 481

Vectoriser 451

Vélocité 18, 53

**Version lecteur Flash,
pénétration (statistiques) 287****Vidéo 137, 177**

3D 268

accélérer 204

Adobe After Effects

manuel d'apprentissage 153

Mode de fusion des calques
198

relief 377

Adobe F4V 162

Adobe Media Encoder

Autres 168

FTP 168

Passes d'encodage 165

Réglages avancés 166

Réglages de débit

CBR, VBR 165

Variabilité du débit 165

affichage en relief 377

agrandir 197

Apple Motion 165

arrêter 204, 232, 235

dans un SWF imbriqué 191

depuis le document racine
234

Audio 183

via FLV 161, 167

boucle 215

bruit 139

cadence 138, 143

capture 139

chapitrage 207, 211, 218

codage différentiel 180

codec 137

compatibilité 163

format vidéo de Flash 140

composant FLVPlayBack Voir
FLVPlayBack

composite dans Flash 169

compression 140, 152, 166

configuration utilisateur 140,
141

débit 151, 165

détecter la fin 217

détournage du fond vert 153

dimensions 268

standard 164

effet 3D 174

enchaîner les vidéos 217

encodage 217, 219

exporter pour Flash 6 191

externaliser 139

F4V 177, 179, 218, 224, 227

Flash Media Server 210

flux simultanés 139

FLV 162, 218, 224, 227-228

générateur de particules 141

HD 139

images-clés 166, 204, 220

intégrée 229

problème de
synchronisation 165

intégrer 139

interactive 201

lecture automatique 203

lire 204

avec des boutons
préprogrammés 172
en arrière (vitesse variable et
fluide) 227

lissage 268

lumière (prise de vue) 152

métadonnées 226

mise en cache 139, 203

Points de repère (CuePoints)

dynamiques (en
ActionScript) 226, 227
événement 217, 223
navigation 217, 220

profondeur de couleur 143

qualité et échelle 139

ratio des pixels 138, 143

remappage temporel 165

rembobiner 205

rollOver 276

source 208

sous-titrage 211

activer/désactiver 215

standard 139

streaming 210

timecode 221, 226

trame 197

entrelacement, progressif
138

transparence 137, 163

voir aussi Adobe After Effects,
Adobe Media Encoder,
Adobe Premiere Pro, Apple
Final Cut Pro, Apple iMovie,
Apple Motion, Cadence, On2
VP6, Quick Time, Sorenson
Spark, Window Movie Maker
volume audio 205, 206**videoEvent 215****Viewport3D 308****Visibilité 396****visible 396**

propriété 12

volume() 205**W****Window Movie Maker 154****wmode 435****X****x, propriété 12****XFL 450****XML 113, 211, 217**

atteindre

attribut 119

nœud 119

cibler un attribut connu 120

créer un document XML [117](#)

importer du HTML [119](#)

length() [119](#)

lire [119](#)

mécanisme [116](#)

nœud [117](#)

optimiser la gestion des
données [121](#)

site dynamique [134](#)

structure [116](#)

XMP [467](#)

Y

y, propriété [12](#)

yaw [310](#)

Z

z, propriété [279](#)

Zone active sur un symbole [48](#)

Zone de défilement [54](#)

Zoom [122](#), [260](#)

Index des notes et encadrés

3D

Accéder directement à une bibliothèque d'objets 3D KMZ [300](#)
 Aide en ligne de PaperVision [328](#)
 Conditions d'utilisation de la banque d'objets 3D Google Sketchup [300](#)
 Gestion des lumières dans PaperVision [328](#)
 Guide de référence PaperVision [328](#)
 Identifier des groupes d'objets pour PaperVision [303](#)
 Interactivité sur les objets 3D [328](#)
 La sous-classe Camera3D [311](#)
 La vidéo en relief [377](#)
 Les primitives de PaperVision [320](#)
 Limite de taille des fichiers 3D [300](#)
 Lisser les images pour la 3D [250](#)
 Optimiser un site 3D [314](#)
 Placer un symbole par-dessus un objet 3D [309](#)
 Réaliser un livre interactif avec InDesign [250](#)
 Sketchup [304](#)
 Vérifier les valeurs de ses propres touches clavier [323](#)

À retenir [14](#), [18](#), [22](#), [28](#), [38](#), [48](#), [55](#), [64](#), [75](#), [85](#), [91](#), [94](#), [97](#), [103](#), [107](#), [113](#), [121](#), [131](#), [134](#), [147](#), [153](#), [169](#), [174](#), [184](#), [188](#), [191](#), [197](#), [199](#), [207](#), [211](#), [215](#), [216](#), [227](#), [231](#), [238](#), [251](#), [260](#), [268](#), [279](#), [287](#), [299](#), [304](#), [311](#), [314](#), [329](#), [335](#), [337](#), [343](#), [353](#), [355](#), [356](#), [360](#), [377](#), [387](#), [395](#), [404](#), [407](#), [414](#), [420](#), [422](#), [427](#), [430](#), [434](#), [435](#), [443](#), [449](#), [451](#), [458](#), [462](#), [464](#), [467](#), [469](#), [476](#), [481](#), [482](#)

Accessibilité

Aides au développement de contenus spécifiques pour les malentendants et les malvoyants [481](#)
 Informations sur l'accessibilité dans Flash [481](#)

Animation

Animation de filtres [353](#)
 Définir les coordonnées d'une courbe de Bézier pour la classe TweenMax [44](#)
 Définition d'une interpolation TweenMax [45](#)
 Définition de la classe transitionManager [62](#)
 Éviter les images tremblantes [27](#)
 Gérer un ascenseur avec une accélération [53](#)
 L'assistant TweenMax [42](#)
 Mise à jour de la classe GreenSock TweenMax [41](#)
 Stabiliser une interpolation Tween [36](#)

Conception et intégration

Dimensions standard d'une vidéo pour le Web [164](#)

Graphisme

Centrer les images avec le composant UI Loader [110](#)
 Définition des paramètres du filtre flou BlurFilter [68](#)
 Définitions des paramètres du filtre biseau BevelFilter [70](#)
 Définitions des paramètres du filtre halo GlowFilter [70](#)
 Définitions des paramètres du filtre ombre portée DropShadowFilter [69](#)
 Faut-il renseigner toutes les propriétés pour les filtres ? [73](#)

Placer un nom d'étiquette sur une image du scénario [426](#)
 Réaliser une image de type motif, raccord, pour panoramiques [24](#)
 Résolution limite des images [349](#)
 Système hexadécimal [356](#)

HTML

Créer un lien HTML sans ActionScript dans Flash [422](#)
 Les cibles d'affichage [421](#)
 Tester les liens HTML sur un serveur distant [427](#)

Langage

3D avec la classe Caurina [250](#)
 3D avec la classe Tween [250](#)
 3D avec la classe TweenMax [250](#)
 Arrêter un écouteur [55](#)
 Arrêter un Timer [64](#)
 Arrondir une valeur [101](#)
 Calcul de la durée d'un chronomètre [59](#)
 Calculer avec la class Math [22](#)
 Calculer un pas d'incrémementation [26](#)
 Ciblage dans un document Flash AS3 [237](#)
 Ciblage des boutons entre SWF imbriqués [415](#)
 Composant FLVPlayback pour ActionScript 2 ou 3 [170](#)
 Conversion de degrés en radians et inversement [22](#)
 Créer un fichier XML [117](#)
 Définir les chemins pour les requêtes externes [102](#)
 Définition du constructeur Point() [84](#)
 Détecter la fin d'une boucle d'itération Timer [64](#)
 Différence entre Loader et URLLoader [119](#)

- Différence entre target et currentTarget 129
- Doit-on toujours placer un écouteur sur un chargeur pour activer l’affichage d’un contenu chargé dynamiquement ? 103
- Enchaîner des actions à une interpolation TweenMax 47
- Exemples de ciblage de contenus dans des SWF imbriqués (AS2 et AS3) 237
- Exporter un symbole pour ActionScript 383
- Générer une valeur aléatoire 101
- L’ordre numérique en ActionScript 112
- Le transtypage 64
- Les classes 11
- Les noms du fichier appelant 95
- Les propriétés en ActionScript 3 12
- Les touches de commande 321
- Lire le XML 119
- Mécanisme d’un tableau (Array) 384
- Mécanisme d’une boucle for 127
- Nommer les occurrences 10
- Nommer une fonction 10
- Options d’impression 354
- Où placer l’écouteur et la fonction ? 63
- Paramétrer un composant via ActionScript 174
- Philosophie de AS3 9
- Pourquoi transtyper un chargeur en MovieClip ? 235
- Propriété overwrite pour TweenMax 413
- Propriétés d’alignement de la scène 446
- Propriétés 3D de la scène 284
- Que faire des classes après compilation des SWF ? 42
- Syntaxe chameau et séparateurs 10
- Target et currentTarget 415
- Typage des nombres 20
- Logiciel**
 - Centraliser les classes ajoutées avec les chemins de classe (classPath) 42
 - Installer une classe importée 428
 - Localiser les classes natives de Flash 32
- Navigation**
 - Différence entre les propriétés alpha et visible 396
 - Menus avec des symboles boutons 395
- PDF**
 - Convertir un PDF en SWF avec FlashPaper 451
 - Création dynamique de fichiers PDF 451
 - Différence structurelle entre Flash et Acrobat 450
- Sites Full Flash**
 - Activer l’accélération matérielle pour les contenus riches 441
 - Désactivation des contrôles du clavier en plein écran 441
 - Modes d’affichage de la scène 445
 - Principe de flottement façon styles CSS 449
 - Quitter le mode Plein écran avec Echap 443
- Squelette**
 - Cibler un segment 84
 - Construire un squelette pour la programmation 80
 - Création de squelettes humains 82
 - Heure de création et exécution 78
- Typographie**
 - Affichage des polices dans Flash 452
 - Le nouveau moteur de texte Text Layout Framework 458
 - Les polices et les masques 454
 - Vectoriser un texte dans Flash 451
- Vidéo**
 - Chapitrage vidéo avec les points de repère 211
 - Comment réactiver les champs de dimensionnement de l’encodeur ? 220
 - Composant FLVPlayback pour ActionScript 2 ou 3 170
 - Conflit entre la cadence des vidéos et la cadence des animations Flash 165
 - Définition des composants associés à FLVPlayback 190
 - Désactiver et réactiver l’affichage des sous-titres 215
 - Enchaîner plusieurs vidéos à la suite 217
 - Encoder en FLV avec Apple Compressor 147
 - Étendre les formatages du document XML pour les sous-titres 215
 - Exporter le projet After Effects natif directement vers Flash 153
 - Flash Media Server 210
 - Icône de sélection de fichier du composant FLVPlayback 171
 - Intégrer physiquement une vidéo dans le scénario 229
 - Le composant CaptionButton 215
 - Les formats pris en charge par Adobe Media Encoder 156
 - Les images-clés 166
 - Mécanisme du codec On2 VP6 162
 - Modifier la cadence des images 165
 - Paramètre source des composants vidéos 172
 - Reconstituer un univers 3D avec une vidéo aplatie 174
 - Trucage avec captation sur fond vert 152
 - Utiliser le FLV pour l’audio, la vidéo ou les deux 161

Le Campus

ActionScript 3 et le motion design

Programmer facilement la vidéo en HD, la 3D, le relief, les Tweens... dans le scénario !

Vous êtes graphiste et débutez en ActionScript 3 ou utilisez déjà ActionScript 2. Vous aimeriez contrôler les contenus et profiter des performances de ce nouveau langage directement dans le scénario de Flash. Ce livre est fait pour vous !

Vous y trouverez des solutions nouvelles, concrètes et clés en main. Vous y acquerez les notions d'ActionScript indispensables pour pouvoir concevoir, sans l'aide d'un développeur, des interfaces sophistiquées et dynamiques à partir de symboles placés dans le scénario.

Sont abordés de façon claire et didactique : la vidéo en qualité HD avec toute l'interactivité, la 3D native, l'animation, les classes d'animation Tweens et TweenMax, les effets et les filtres, la gestion de documents imbriqués désormais plus complexe. Mais aussi de nombreux concepts inédits, comme le référencement de documents Flash, le relief, l'animation de squelettes et la 3D avec PaperVision.

Plus qu'un livre d'apprentissage, cet ouvrage rassemble toutes les techniques qu'un bon web designer doit aujourd'hui connaître. Un outil indispensable en production.

À propos de l'auteur

Arzhur Caouissin, auteur et réalisateur multimédia indépendant, est formateur depuis plus de dix ans à l'école des Gobelins, chez Pyramid et à l'INA. Spécialisé dans la technologie Flash, il est apprécié pour sa vision transversale et artistique des différents médias, et pour son expertise en création de contenus web enrichis.



Codes sources et galerie sur www.pearson.fr !

Table des matières

- Les animations en ActionScript
- Interpolations et interactivité avec les classes Tween et TweenMax
- Les transitions d'effets et de filtres
- La programmation de squelettes
- Les galeries d'images
- La vidéo standard et composite en FLV
- La vidéo HD en F4V
- La vidéo interactive
- La 3D native
- La 3D et PaperVision
- API d'affichage et de colorimétrie
- Le Web en vrai relief
- Les systèmes de navigation avancés
- La communication Flash/HTML
- La gestion de sites Full Flash
- Les solutions de référencement du Flash
- L'accessibilité dans Flash
- Ressources

Niveau : Intermédiaire / Avancé

Catégorie : Graphisme web /

Programmation

**Configuration : Flash CS4 et ultérieur,
Mac OS / Windows**

PEARSON

Pearson Education France
47 bis rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

ISBN : 978-2-7440-4128-0

